



Kinetis SDK v.1.2 API Reference Manual

Freescale Semiconductor, Inc.

Document Number: KSDK12APIRM
Rev. 0
Apr 2015



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	Data Structure Documentation	18
3.2.3	Enumeration Type Documentation	19
3.2.4	Function Documentation	22
3.2.5	Variable Documentation	24
3.3	ADC16 Peripheral Driver	25
3.3.1	Overview	25
3.3.2	ADC16 Driver model building	25
3.3.3	ADC16 Initialization	25
3.3.4	ADC16 Call diagram	25
3.3.5	Enumeration Type Documentation	30
3.3.6	Function Documentation	30
3.3.7	Variable Documentation	35

Chapter Analog Front End (AFE)

4.1	Overview	37
4.2	AFE HAL driver	38
4.2.1	Overview	38
4.2.2	Data Structure Documentation	40
4.2.3	Enumeration Type Documentation	42
4.2.4	Function Documentation	43
4.3	AFE Peripheral driver	53
4.3.1	Overview	53
4.3.2	Channel configuration structures	53
4.3.3	User configuration structures	53

Contents

Section Number	Title	Page Number
4.3.4	Initialization	53
4.3.5	Measuring	55
4.3.6	Data Structure Documentation	57
4.3.7	Enumeration Type Documentation	59
4.3.8	Function Documentation	60
4.3.9	Variable Documentation	65
Chapter	Crossbar AND/OR/INVERT (AOI) Module	
5.1	Overview	67
5.2	AOI HAL Driver	68
5.2.1	Overview	68
5.2.2	Enumeration Type Documentation	69
5.2.3	Function Documentation	70
5.3	AOI Peripheral Driver	72
5.3.1	Overview	72
5.3.2	Overview	72
5.3.3	Initialization	72
5.3.4	Event output configuration	72
5.3.5	Call diagram	72
5.3.6	Data Structure Documentation	74
5.3.7	Function Documentation	76
5.3.8	Variable Documentation	79
Chapter	Comparator (CMP)	
6.1	Overview	81
6.2	CMP HAL Driver	82
6.2.1	Overview	82
6.2.2	Data Structure Documentation	84
6.2.3	Enumeration Type Documentation	85
6.2.4	Function Documentation	87
6.3	CMP Peripheral Driver	91
6.3.1	Overview	91
6.3.2	CMP Driver model building	91
6.3.3	CMP Call diagram	91
6.3.4	Data Structure Documentation	94
6.3.5	Enumeration Type Documentation	95
6.3.6	Function Documentation	95
6.3.7	Variable Documentation	99

Contents

Section Number	Title	Page Number
Chapter	Computer Operating Properly (COP) Watchdog Timer	
7.1	Overview	101
7.2	COP HAL driver	102
7.2.1	Overview	102
7.2.2	Data Structure Documentation	103
7.2.3	Enumeration Type Documentation	103
7.2.4	Function Documentation	104
7.3	COP Peripheral Driver	107
7.3.1	Overview	107
7.3.2	COP Initialization	107
7.3.3	COP Refresh	107
7.3.4	COP Reset Count	107
7.3.5	Function Documentation	108
7.3.6	Variable Documentation	110
Chapter	12-bit Cyclic Analog-to-Digital Converter (CADC)	
8.1	Overview	111
8.2	CADC HAL Driver	112
8.2.1	Overview	112
8.2.2	Data Structure Documentation	114
8.2.3	Enumeration Type Documentation	116
8.2.4	Function Documentation	118
8.3	CADC Peripheral Driver	131
8.3.1	Overview	131
8.3.2	CADC Driver model building	131
8.3.3	CADC Work mode	131
8.3.4	CADC Interrupt	132
8.3.5	CADC Call diagram	132
8.3.6	Enumeration Type Documentation	136
8.3.7	Function Documentation	137
8.3.8	Variable Documentation	142
Chapter	Digital-to-Analog Converter (DAC)	
9.1	Overview	143
9.2	DAC HAL Driver	144
9.2.1	Overview	144
9.2.2	Data Structure Documentation	145
9.2.3	Enumeration Type Documentation	146

Contents

Section Number	Title	Page Number
9.2.4	Function Documentation	147
9.3	DAC Peripheral Driver	151
9.3.1	Overview	151
9.3.2	DAC Initialization	151
9.3.3	DAC Model building	151
9.3.4	DAC Call diagram	152
9.3.5	Enumeration Type Documentation	159
9.3.6	Function Documentation	159
9.3.7	Variable Documentation	163
Chapter	Direct Memory Access (DMA)	
10.1	Overview	165
10.2	DMA HAL driver	166
10.2.1	Overview	166
10.2.2	Data Structure Documentation	168
10.2.3	Enumeration Type Documentation	168
10.2.4	Function Documentation	169
10.3	DMAMUX HAL driver	178
10.4	DMA driver	179
10.4.1	Overview	179
10.4.2	Data Structure Documentation	180
10.4.3	Typedef Documentation	181
10.4.4	Enumeration Type Documentation	181
10.4.5	Function Documentation	181
10.4.6	Variable Documentation	185
10.5	DMA request	186
10.5.1	DMA Initialization	186
10.5.2	DMA Channel concept	186
10.5.3	DMA request concept	186
10.5.4	DMA Memory allocation	186
10.5.5	DMA Call diagram	186
Chapter	Serial Peripheral Interface (DSPI)	
11.1	Overview	189
11.2	DSPI HAL driver	190
11.2.1	Overview	190
11.2.2	Data Structure Documentation	194
11.2.3	Enumeration Type Documentation	195

Contents

Section Number	Title	Page Number
11.2.4	Function Documentation	198
11.3	DSPI Master Driver	214
11.3.1	Overview	214
11.3.2	DSPI Introduction	214
11.3.3	DSPI Run-time state structures	215
11.3.4	DSPI User configuration structures	215
11.3.5	DSPI Device structures	216
11.3.6	DSPI Initialization	216
11.3.7	DSPI Transfers	218
11.3.8	DSPI De-initialization	220
11.3.9	Data Structure Documentation	223
11.3.10	Function Documentation	227
11.3.11	Variable Documentation	238
11.4	DSPI Slave Driver	239
11.4.1	Overview	239
11.4.2	DSPI Runtime state of the DSPI slave driver	240
11.4.3	DSPI User configuration structures	240
11.4.4	DSPI Setup and Initialization	240
11.4.5	DSPI Blocking and non-blocking	241
11.4.6	DSPI De-initialization	242
11.4.7	Data Structure Documentation	245
11.4.8	Function Documentation	247
11.4.9	Variable Documentation	253
11.5	DSPI Shared IRQ Driver	254
11.5.1	Overview	254
11.5.2	Function Documentation	254
11.5.3	Variable Documentation	256
Chapter	Enhanced Direct Memory Access (eDMA)	
12.1	Overview	257
12.2	eDMA HAL driver	258
12.2.1	Overview	258
12.2.2	Data Structure Documentation	262
12.2.3	Enumeration Type Documentation	265
12.2.4	Function Documentation	266
12.3	eDMA Peripheral driver	292
12.3.1	Overview	292
12.3.2	eDMA Initialization	292
12.3.3	eDMA Channel concept	292

Contents

Section Number	Title	Page Number
12.3.4	DMA request concept	292
12.3.5	eDMA Memory allocation and alignment	292
12.3.6	eDMA Call diagram	293
12.3.7	eDMA Extend the driver	294
12.3.8	Data Structure Documentation	297
12.3.9	Macro Definition Documentation	298
12.3.10	Typedef Documentation	299
12.3.11	Enumeration Type Documentation	299
12.3.12	Function Documentation	300
12.3.13	Variable Documentation	311
12.4	eDMA request	312
12.4.1	Overview	312
12.4.2	Enumeration Type Documentation	312
Chapter	Quadrature Encoder/Decoder (ENC)	
13.1	Overview	313
13.2	ENC HAL driver	314
13.2.1	Overview	314
13.2.2	Enumeration Type Documentation	318
13.2.3	Function Documentation	319
13.3	ENC Peripheral Driver	354
13.3.1	Overview	354
13.3.2	Overview	354
13.3.3	Initialization	354
13.3.4	Testing ENC module	355
13.3.5	Input monitor	355
13.3.6	Interrupts	356
13.3.7	Reading Counters	356
13.3.8	Reading Hold Registers	356
13.3.9	Data Structure Documentation	357
13.3.10	Function Documentation	360
Chapter	Ethernet MAC (ENET)	
14.1	Overview	365
14.2	ENET HAL driver	366
14.2.1	Overview	366
14.2.2	Data Structure Documentation	373
14.2.3	Macro Definition Documentation	380
14.2.4	Enumeration Type Documentation	380

Contents

Section Number	Title	Page Number
14.2.5	Function Documentation	387
14.3	ENET RTCS Adaptor	398
14.4	ENET Physical Layer Driver	399
14.5	ENET Peripheral Driver	400
14.5.1	Overview	400
14.5.2	ENET device data structure	400
14.5.3	ENET Configuration structure	400
14.5.4	ENET Buffer data structure	400
14.5.5	ENET Initialization	401
14.5.6	ENET Data Receive	402
14.5.7	ENET Data Transmit	407
14.5.8	ENET Note:	409
14.5.9	Data Structure Documentation	411
14.5.10	Macro Definition Documentation	415
14.5.11	Enumeration Type Documentation	415
14.5.12	Function Documentation	415
14.5.13	Variable Documentation	421
 Chapter External Watchdog Timer (EWM)		
15.1	Overview	423
15.2	EWM HAL driver	424
15.2.1	Overview	424
15.2.2	Data Structure Documentation	425
15.2.3	Enumeration Type Documentation	425
15.2.4	Function Documentation	425
15.3	EWM Peripheral Driver	429
15.3.1	Overview	429
15.3.2	EWM Initialization	429
15.3.3	EWM Refresh	429
15.3.4	Function Documentation	430
15.3.5	Variable Documentation	431
 Chapter C90TFS Flash Driver		
16.1	Overview	433
16.2	Data Structure Documentation	441
16.2.1	struct FLASH_SSD_CONFIG	441
16.3	Macro Definition Documentation	442

Contents

Section Number	Title	Page Number
16.3.1	BYTE2WORD	442
16.3.2	WORD2BYTE	442
16.3.3	SET_FLASH_INT_BITS	442
16.3.4	GET_FLASH_INT_BITS	442
16.3.5	FTFx_ERR_MGSTAT0	442
16.3.6	FTFx_ERR_PVIOL	443
16.3.7	FTFx_ERR_ACCERR	443
16.3.8	FTFx_ERR_CHANGEPROT	443
16.3.9	FTFx_ERR_NOEEE	443
16.3.10	FTFx_ERR_EFLASHONLY	444
16.3.11	FTFx_ERR_RAMRDY	444
16.3.12	FTFx_ERR_RANGE	444
16.3.13	FTFx_ERR_SIZE	444
16.4	Function Documentation	444
16.4.1	RelocateFunction	444
16.4.2	FlashInit	445
16.4.3	FlashCommandSequence	445
16.4.4	PFlashGetProtection	446
16.4.5	PFlashSetProtection	446
16.4.6	FlashGetSecurityState	447
16.4.7	FlashSecurityBypass	447
16.4.8	FlashEraseAllBlock	448
16.4.9	FlashVerifyAllBlock	449
16.4.10	FlashEraseSector	449
16.4.11	FlashVerifySection	450
16.4.12	FlashEraseSuspend	451
16.4.13	FlashEraseResume	452
16.4.14	FlashReadOnce	452
16.4.15	FlashProgramOnce	453
16.4.16	FlashReadResource	454
16.4.17	FlashProgram	455
16.4.18	FlashProgramCheck	455
16.4.19	FlashCheckSum	456
16.4.20	FlashProgramSection	457
16.4.21	FlashEraseBlock	457
16.4.22	FlashVerifyBlock	458
Chapter	External Bus Interface (FLEXBUS)	
17.1	Overview	459
17.2	FLEXBUS HAL driver	460
17.2.1	Overview	460
17.2.2	Data Structure Documentation	463

Contents

Section Number	Title	Page Number
17.2.3	Enumeration Type Documentation	464
17.2.4	Function Documentation	466
17.3	FLEXBUS Peripheral Driver	479
17.3.1	Overview	479
17.3.2	FLEXBUS Overview	479
17.3.3	FLEXBUS Initialization	479
17.3.4	FLEXBUS De-initialize	479
17.3.5	Function Documentation	480
17.3.6	Variable Documentation	480
Chapter	Controller Area Network (FlexCAN)	
18.1	Overview	481
18.2	FlexCAN HAL driver	482
18.2.1	Overview	482
18.2.2	Data Structure Documentation	486
18.2.3	Enumeration Type Documentation	488
18.2.4	Function Documentation	491
18.3	FlexCAN Driver	507
18.3.1	Overview	507
18.3.2	FlexCAN Overview	507
18.3.3	FlexCAN Initialization	507
18.3.4	FlexCAN Module timing	507
18.3.5	FlexCAN Transfers	508
18.3.6	Data Structure Documentation	510
18.3.7	Function Documentation	512
18.3.8	Variable Documentation	520
Chapter	FlexTimer (FTM)	
19.1	Overview	521
19.2	FlexTimer HAL driver	522
19.2.1	Overview	522
19.2.2	Data Structure Documentation	528
19.2.3	Macro Definition Documentation	529
19.2.4	Enumeration Type Documentation	529
19.2.5	Function Documentation	530
19.3	FlexTimer Peripheral Driver	561
19.3.1	Overview	561
19.3.2	FlexTimer Overview	561

Contents

Section Number	Title	Page Number
19.3.3	FlexTimer Initialization	561
19.3.4	FlexTimer Generate a PWM signal	561
19.3.5	Data Structure Documentation	563
19.3.6	Function Documentation	563
19.3.7	Variable Documentation	568
Chapter	General Purpose Input/Output (GPIO)	
20.1	Overview	569
20.2	GPIO HAL driver	570
20.2.1	Overview	570
20.2.2	Enumeration Type Documentation	571
20.2.3	Function Documentation	571
20.3	GPIO Peripheral driver	580
20.3.1	Overview	580
20.3.2	GPIO Pin Definitions	580
20.3.3	GPIO Initialization	580
20.3.4	Output Operations	581
20.3.5	Input Operations	582
20.3.6	GPIO Interrupt	582
20.3.7	Data Structure Documentation	584
20.3.8	Macro Definition Documentation	585
20.3.9	Function Documentation	585
20.3.10	Variable Documentation	590
Chapter	Inter-Integrated Circuit (I2C)	
21.1	Overview	591
21.2	I2C HAL driver	592
21.2.1	Overview	592
21.2.2	I2C ICR Table	592
21.2.3	I2C Clock rate formulas	592
21.2.4	Enumeration Type Documentation	595
21.2.5	Function Documentation	596
21.3	I2C Slave peripheral driver	609
21.3.1	Overview	609
21.3.2	Data Structure Documentation	610
21.3.3	Enumeration Type Documentation	612
21.3.4	Function Documentation	613
21.3.5	Variable Documentation	619

Contents

Section Number	Title	Page Number
21.4	I2C Master peripheral	620
21.4.1	Overview	620
21.4.2	I2C Initialization	620
21.4.3	I2C Data Transactions	620
21.4.4	Data Structure Documentation	622
21.4.5	Function Documentation	622
21.4.6	Variable Documentation	628
Chapter	Independent Real Time Clock (IRTC)	
22.1	Overview	629
22.2	IRTC HAL driver	630
22.2.1	IRTC Initialization	630
22.2.2	IRTC Setting and reading the IRTC time	630
22.2.3	IRTC Setting and reading the Alarm	630
22.3	IRTC Peripheral Driver	631
22.3.1	Overview	631
22.3.2	IRTC Peripheral Driver Initialization	631
22.3.3	IRTCS Setting and reading the IRTC time	631
22.3.4	IRTC Triggering an Alarm	631
22.3.5	Function Documentation	633
22.3.6	Variable Documentation	636
Chapter	Local Memory Controller (LMEM) Cache Control Driver	
23.1	Overview	637
23.2	LMEM Cache HAL driver	638
23.2.1	Overview	638
23.2.2	Macro Definition Documentation	639
23.2.3	Enumeration Type Documentation	640
23.2.4	Function Documentation	641
23.3	LMEM Cache Driver	646
23.3.1	Overview	646
23.3.2	LMEM Cache Driver Definitions and Usage	646
23.3.3	Function Documentation	648
23.3.4	Variable Documentation	652
Chapter	Universal Asynchronous Receiver/Transmitter (LPSCI)	
24.1	Overview	653

Contents

Section Number	Title	Page Number
24.2	LPSCI HAL driver	654
24.2.1	Overview	654
24.2.2	Enumeration Type Documentation	658
24.2.3	Function Documentation	661
24.3	LPSCI Peripheral driver	674
24.3.1	Overview	674
24.3.2	LPSCI Device structures	674
24.3.3	LPSCI User configuration structures	674
24.3.4	LPSCI Initialization	674
24.3.5	LPSCI Transfers	675
24.3.6	Data Structure Documentation	678
24.3.7	Typedef Documentation	682
24.3.8	Function Documentation	682
24.3.9	Variable Documentation	692
Chapter	Low Power Timer (LPTMR)	
25.1	Overview	693
25.2	LPTMR HAL driver	694
25.2.1	Overview	694
25.2.2	Data Structure Documentation	696
25.2.3	Enumeration Type Documentation	697
25.2.4	Function Documentation	698
25.3	LPTMR Peripheral driver	702
25.3.1	Overview	702
25.3.2	LPTMR Initialization	702
25.3.3	LPTMR Interrupt	702
25.3.4	Data Structure Documentation	704
25.3.5	Function Documentation	705
25.3.6	Variable Documentation	708
Chapter	Low Power Universal Asynchronous Receiver/Transmitter (LPUART)	
26.1	Overview	709
26.2	LPUART HAL driver	710
26.2.1	Overview	710
26.2.2	Data Structure Documentation	715
26.2.3	Enumeration Type Documentation	716
26.2.4	Function Documentation	719
26.3	LPUART Peripheral driver	735

Contents

Section Number	Title	Page Number
26.3.1	Overview	735
26.3.2	Data Structure Documentation	737
26.3.3	Typedef Documentation	742
26.3.4	Function Documentation	742
26.3.5	Variable Documentation	756
26.4	LPUART Type Definitions	757
26.4.1	LPUART Overview	757
26.4.2	LPUART Device structures	757
26.4.3	LPUART Initialization	757
26.4.4	LPUART Transfers	757
Chapter	Memory Protection Unit (MPU)	
27.1	Overview	759
27.2	MPU HAL driver	760
27.2.1	Overview	760
27.2.2	Data Structure Documentation	762
27.2.3	Enumeration Type Documentation	764
27.2.4	Function Documentation	766
27.3	MPU Peripheral driver	770
27.3.1	Overview	770
27.3.2	MPU Initialization	770
27.3.3	MPU Interrupt	771
27.3.4	Data Structure Documentation	772
27.3.5	Function Documentation	772
27.3.6	Variable Documentation	775
Chapter	Programmable Delay Block (PDB)	
28.1	Overview	777
28.2	PDB HAL driver	778
28.2.1	Overview	778
28.2.2	Data Structure Documentation	780
28.2.3	Enumeration Type Documentation	781
28.2.4	Function Documentation	783
28.3	PDB Peripheral driver	792
28.3.1	Overview	792
28.3.2	PDB Driver model building	792
28.3.3	PDB Initialization	792
28.3.4	PDB Call diagram	792

Contents

Section Number	Title	Page Number
28.3.5	Data Structure Documentation	796
28.3.6	Function Documentation	797
28.3.7	Variable Documentation	804
Chapter	Periodic Interrupt Timer (PIT)	
29.1	Overview	805
29.2	PIT HAL driver	806
29.2.1	Overview	806
29.2.2	Enumeration Type Documentation	807
29.2.3	Function Documentation	807
29.3	PIT Peripheral driver	812
29.3.1	Overview	812
29.3.2	PIT Initialization	812
29.3.3	PIT Timer Period	812
29.3.4	PIT Timer Operation	813
29.3.5	Data Structure Documentation	814
29.3.6	Function Documentation	814
29.3.7	Variable Documentation	819
Chapter	Pulse Width Modulator A (PWMA/eFlexPWM)	
30.1	Overview	821
30.2	eFlexPWM Peripheral Driver	822
30.2.1	Overview	822
30.2.2	Initialization	822
30.2.3	Generate a PWM signal	822
30.2.4	Data Structure Documentation	824
30.2.5	Enumeration Type Documentation	824
30.2.6	Function Documentation	825
30.2.7	Variable Documentation	830
Chapter	FlexTimer (FTM)	
31.1	Overview	831
31.2	FlexTimer HAL driver	832
31.2.1	Overview	832
31.2.2	Data Structure Documentation	834
31.2.3	Enumeration Type Documentation	835
31.2.4	Function Documentation	836

Contents

Section Number	Title	Page Number
31.3	FlexTimer Peripheral Driver	844
31.3.1	Overview	844
31.3.2	FlexTimer Overview	844
31.3.3	FlexTimer Initialization	844
31.3.4	FlexTimer Generate a PWM signal	844
31.3.5	Enumeration Type Documentation	846
31.3.6	Function Documentation	846
31.3.7	Variable Documentation	848
Chapter	Random Number Generator Accelerator (RNGA)	
32.1	Overview	849
32.2	RNGA HAL driver	850
32.2.1	Overview	850
32.2.2	Enumeration Type Documentation	851
32.2.3	Function Documentation	852
32.3	RNGA Peripheral Driver	859
32.3.1	Overview	859
32.3.2	RNGA Initialization	859
32.3.3	RNGA Set/Get Working Mode	859
32.3.4	Get random data from RNGA	859
32.3.5	Seed RNGA	859
32.3.6	RNGA interrupt	859
32.3.7	Data Structure Documentation	860
32.3.8	Function Documentation	861
32.3.9	Variable Documentation	863
Chapter	Real Time Clock (RTC)	
33.1	Overview	865
33.2	RTC HAL driver	866
33.2.1	Overview	866
33.2.2	RTC Initialization	866
33.2.3	RTC Setting and reading the RTC time	866
33.2.4	IRTC Setting and reading the Alarm	866
33.2.5	Data Structure Documentation	872
33.2.6	Enumeration Type Documentation	874
33.2.7	Function Documentation	875
33.3	RTC Peripheral Driver	903
33.3.1	Overview	903
33.3.2	RTC Peripheral Driver Initialization	903

Contents

Section Number	Title	Page Number
33.3.3	IRTC Setting and reading the IRTC time	903
33.3.4	RTC Triggering an Alarm	903
33.3.5	RTC Repeated alarm	904
33.3.6	RTC Enable and Disable Alarm Interrupts	905
33.3.7	RTC Interrupt handler	905
33.3.8	Data Structure Documentation	907
33.3.9	Function Documentation	907
33.3.10	Variable Documentation	913
Chapter	Synchronous Audio Interface (SAI)	
34.1	Overview	915
34.2	SAI HAL driver	916
34.2.1	Overview	916
34.2.2	Data Structure Documentation	919
34.2.3	Enumeration Type Documentation	919
34.2.4	Function Documentation	922
34.3	SAI Peripheral driver	931
34.3.1	Overview	931
34.3.2	SAI Initialization	931
34.3.3	SAI Configuration	931
34.3.4	SAI Call diagram	931
34.3.5	Data Structure Documentation	934
34.3.6	Function Documentation	935
Chapter	Secured Digital Host Controller (SDHC)	
35.1	Overview	945
35.2	SDHC HAL	946
35.2.1	Overview	946
35.2.2	Data Structure Documentation	948
35.2.3	Enumeration Type Documentation	950
35.2.4	Function Documentation	951
35.3	SDHC Peripheral Driver	957
35.3.1	Overview	957
35.3.2	SDHC Initialization	957
35.3.3	SDHC Issuing a request to the card	957
35.3.4	Function Documentation	958
35.4	SDHC Data Types	961
35.4.1	Overview	961

Contents

Section Number	Title	Page Number
35.4.2	Data Structure Documentation	963
35.4.3	Enumeration Type Documentation	966
35.5	SDHC Standard Definition	969
35.5.1	Overview	969
35.6	SDHC Card Related Standard Definition	978
35.7	SDHC Card Definition	979
35.8	SDHC Card Driver	980
35.9	SD Card SPI Data Definition	981
35.10	SD Card SPI Driver	982
Chapter	LCD (SLCD)	
36.1	Overview	983
36.2	SLCD HAL driver	984
36.2.1	Overview	984
36.2.2	Data Structure Documentation	988
36.2.3	Enumeration Type Documentation	989
36.2.4	Function Documentation	992
36.3	SLCD Peripheral driver	1000
36.3.1	Overview	1000
36.3.2	Initialization	1000
36.3.3	Data Structure Documentation	1002
36.3.4	Function Documentation	1002
36.3.5	Variable Documentation	1007
Chapter	Serial Peripheral Interface (SPI)	
37.1	Overview	1009
37.2	SPI HAL driver	1010
37.2.1	Overview	1010
37.2.2	Enumeration Type Documentation	1014
37.2.3	Function Documentation	1017
37.3	SPI Master Peripheral Driver	1026
37.3.1	Overview	1026
37.3.2	SPI Run-time state structures	1027
37.3.3	SPI Device structures	1027

Contents

Section Number	Title	Page Number
37.3.4	SPI Initialization1027
37.3.5	SPI Transfers1029
37.3.6	SPI De-initialization1030
37.3.7	Data Structure Documentation1033
37.3.8	Enumeration Type Documentation1035
37.3.9	Function Documentation1036
37.3.10	Variable Documentation1044
37.4	SPI Slave Peripheral Driver1045
37.4.1	Overview1045
37.4.2	SPI Overview1045
37.4.3	SPI Runtime state of the SPI slave driver1046
37.4.4	SPI User configuration structures1046
37.4.5	SPI Setup and Initialization1046
37.4.6	SPI Blocking and non-blocking1047
37.4.7	SPI De-initialization1048
37.4.8	Data Structure Documentation1050
37.4.9	Function Documentation1053
37.4.10	Variable Documentation1060
37.5	Shared SPI Types1061
37.6	SPI Classes1062
Chapter	Timer/PWM Module (TPM)	
38.1	Overview1063
38.2	TPM HAL driver1064
38.2.1	Overview1064
38.2.2	Data Structure Documentation1067
38.2.3	Enumeration Type Documentation1067
38.2.4	Function Documentation1068
38.3	TPM Peripheral driver1079
38.3.1	Overview1079
38.3.2	TPM Initialization1079
38.3.3	TPM Generate a PWM signal1079
38.3.4	TPM Input Capture1079
38.3.5	TPM Output Compare1080
38.3.6	TPM Interrupt handler1080
38.3.7	Data Structure Documentation1082
38.3.8	Enumeration Type Documentation1082
38.3.9	Function Documentation1083
38.3.10	Variable Documentation1088

Contents

Section Number	Title	Page Number
Chapter	Touch Sense Input (TSI)	
39.1	Overview	1089
39.2	TSI HAL driver	1090
39.2.1	Overview1090
39.2.2	Data Structure Documentation1100
39.2.3	Enumeration Type Documentation1102
39.2.4	Function Documentation1109
39.3	TSI Peripheral Driver	1138
39.3.1	Overview1138
39.3.2	Overview1138
39.3.3	TSI Initialization1138
39.3.4	Data Structure Documentation1142
39.3.5	Typedef Documentation1144
39.3.6	Enumeration Type Documentation1144
39.3.7	Function Documentation1145
39.3.8	Variable Documentation1153
Chapter	Universal Asynchronous Receiver/Transmitter (UART)	
40.1	Overview	1155
40.2	UART HAL driver	1156
40.2.1	Overview1156
40.2.2	Enumeration Type Documentation1160
40.2.3	Function Documentation1164
40.3	UART Peripheral driver	1177
40.3.1	Overview1177
40.3.2	UART Device structures1177
40.3.3	UART User configuration structures1177
40.3.4	UART Initialization1177
40.3.5	UART Transfers1178
40.3.6	Data Structure Documentation1180
40.3.7	Function Documentation1183
40.3.8	Variable Documentation1189
Chapter	Reference (VREF)	
41.1	Overview	1191
41.2	VREF HAL Driver	1192
41.2.1	Overview1192
41.2.2	Data Structure Documentation1193

Contents

Section Number	Title	Page Number
41.2.3	Enumeration Type Documentation1193
41.2.4	Function Documentation1193
41.3	VREF Peripheral driver1197
41.3.1	Overview1197
41.3.2	VREF Initialization1197
41.3.3	Function Documentation1198
 Chapter Watchdog Timer (WDOG)		
42.1	Overview1201
42.2	WDOG HAL driver1202
42.2.1	Overview1202
42.2.2	Data Structure Documentation1203
42.2.3	Enumeration Type Documentation1204
42.2.4	Function Documentation1205
42.3	WDOG Peripheral driver1210
42.3.1	Overview1210
42.3.2	WDOG Initialization1210
42.3.3	WDOG Refresh1210
42.3.4	WDOG Reset Count1210
42.3.5	WDOG Reset System1210
42.3.6	WDOG interrupt1211
42.3.7	Function Documentation1211
 Chapter Inter-Peripheral Crossbar Switch (XBAR)		
43.1	Overview1213
43.2	XBAR HAL driver1214
43.2.1	Overview1214
43.2.2	Enumeration Type Documentation1215
43.2.3	Function Documentation1215
43.3	XBAR Peripheral driver1221
43.3.1	Overview1221
43.3.2	Overview1221
43.3.3	CMP Driver model building1221
43.3.4	Initialization1221
43.3.5	Call diagram1221
43.3.6	Data Structure Documentation1225
43.3.7	Typedef Documentation1226
43.3.8	Enumeration Type Documentation1226

Contents

Section Number	Title	Page Number
43.3.9	Function Documentation1226
43.3.10	Variable Documentation1229
 Chapter Low-Leakage Wakeup Unit (LLWU)		
44.1	Overview1231
44.2	External Wakeup Pin APIs1231
44.3	Internal Wakeup Module APIs1231
44.4	Reset Pin Configure APIs1231
44.5	Data Structure Documentation1232
44.5.1	struct llwu_external_pin_filter_mode_t1232
44.5.2	struct llwu_reset_pin_mode_t1232
44.6	Enumeration Type Documentation1232
44.6.1	llwu_external_pin_modes_t1232
44.6.2	llwu_filter_modes_t1233
 Chapter Multipurpose Clock Generator (MCG)		
45.1	Overview1235
45.2	MCG HAL driver1236
45.2.1	Overview1236
45.2.2	OSC related APIs1236
45.2.3	FLL reference clock source, divider, and output frequency.1236
45.2.4	PLL reference clock source, divider and output frequency1237
45.2.5	MCG mode related APIs1237
45.2.6	MCG auto trim machine(ATM) function1237
45.2.7	Enumeration Type Documentation1243
45.2.8	Function Documentation1247
 Chapter Multipurpose Clock Generator Lite (MCG_Lite)		
46.1	Overview1267
46.2	MCG_Lite HAL driver1268
46.2.1	Overview1268
46.2.2	Get current clock output1268
46.2.3	Clock mode switching1268
46.2.4	Enumeration Type Documentation1271
46.2.5	Function Documentation1273

Contents

Section Number	Title	Page Number
Chapter	Memory-Mapped Divide and Square Root(MMDVSQ)	
47.1	Overview1281
47.2	MMDVSQ HAL driver1282
47.2.1	Overview1282
47.2.2	Perform MMDVSQ Divide or square root operation1282
47.2.3	Function Documentation1284
Chapter	Oscillator (OSC)	
48.1	Overview1295
48.2	OSC HAL driver1296
48.2.1	Overview1296
48.2.2	OSCERCLK Configure APIs1296
48.2.3	Capacitor Load Configure APIs1296
48.2.4	Enumeration Type Documentation1297
48.2.5	Function Documentation1297
Chapter	Power Management Controller (PMC)	
49.1	Overview1299
49.2	PMC HAL driver1300
49.2.1	Overview1300
49.2.2	Data Structure Documentation1301
49.2.3	Enumeration Type Documentation1301
49.2.4	Function Documentation1302
Chapter	Port Control and Interrupts (PORT)	
50.1	Overview1307
50.2	PORT HAL driver1308
50.2.1	Overview1308
50.2.2	Enumeration Type Documentation1309
50.2.3	Function Documentation1310
Chapter	Reset Control Module (RCM)	
51.1	Overview1315
51.2	RCM HAL driver1316
51.2.1	Overview1316

Contents

Section Number	Title	Page Number
51.2.2	System Reset Source APIs1316
51.2.3	RESET Pin Filter APIs1316
51.2.4	ROM Boot APIs1316
51.2.5	Data Structure Documentation1317
51.2.6	Enumeration Type Documentation1318
51.2.7	Function Documentation1318
Chapter	System Clock Generator (SCG)	
52.1	Overview1321
52.2	SCG HAL driver1322
52.2.1	APIs for MCU system clock1322
52.2.2	APIs for clock out selection1322
52.2.3	APIs for clock source configuration1322
Chapter	System Integration Module (SIM)	
53.1	Overview1325
53.2	SIM HAL driver1326
53.2.1	Overview1326
53.2.2	Clock Gate Control register access APIs1326
53.2.3	Clock Source Control access APIs1326
53.2.4	Clock Divider access APIs1327
53.2.5	Macro Definition Documentation1358
53.2.6	Enumeration Type Documentation1358
53.2.7	Function Documentation1377
53.2.8	K02F12810 SIM HAL driver1435
53.2.9	K22F12810 SIM HAL driver1442
53.2.10	K22F25612 SIM HAL driver1452
53.2.11	K22F51212 SIM HAL driver1462
53.2.12	K10D10 SIM HAL driver1472
53.2.13	K11DA5 SIM HAL driver1482
53.2.14	K20D10 SIM HAL driver1490
53.2.15	K21DA5 SIM HAL driver1500
53.2.16	KL02Z4 SIM HAL driver1508
53.2.17	KL03Z4 SIM HAL driver1513
53.2.18	KL13Z644 SIM HAL driver1519
53.2.19	KL16Z4 SIM HAL driver1520
53.2.20	KL17Z4 SIM HAL driver1527
53.2.21	KL17Z644 SIM HAL driver1533
53.2.22	K21FA12 SIM HAL driver1539
53.2.23	K24F12 SIM HAL driver1548
53.2.24	KL26Z4 SIM HAL driver1557

Contents

Section Number	Title	Page Number
53.2.25	KL27Z4 SIM HAL driver1564
53.2.26	KL27Z644 SIM HAL driver1570
53.2.27	KL28T7 SIM HAL driver1576
53.2.28	KL33Z4 SIM HAL driver1584
53.2.29	KL33Z644 SIM HAL driver1590
53.2.30	KL34Z4 SIM HAL driver1591
53.2.31	KL36Z4 SIM HAL driver1598
53.2.32	KL46Z4 SIM HAL driver1605
53.2.33	K24F25612 SIM HAL driver1612
53.2.34	K26F18 SIM HAL driver1621
53.2.35	K30D10 SIM HAL driver1633
53.2.36	K40D10 SIM HAL driver1643
53.2.37	K50D10 SIM HAL driver1653
53.2.38	K51D10 SIM HAL driver1663
53.2.39	K52D10 SIM HAL driver1673
53.2.40	K53D10 SIM HAL driver1683
53.2.41	KV10Z7 SIM HAL driver1693
53.2.42	K60D10 SIM HAL driver1699
53.2.43	KV30F12810 SIM HAL driver1709
53.2.44	KV31F12810 SIM HAL driver1715
53.2.45	KV31F25612 SIM HAL driver1722
53.2.46	KV31F51212 SIM HAL driver1729
53.2.47	K63F12 SIM HAL driver1768
53.2.48	K64F12 SIM HAL driver1778
53.2.49	K65F18 SIM HAL driver1788
53.2.50	K66F18 SIM HAL driver1800
53.2.51	KL43Z4 SIM HAL driver1812
53.2.52	KW01Z4 SIM HAL driver1818
53.2.53	KW21D5 SIM HAL driver1825
53.2.54	KW22D5 SIM HAL driver1833
53.2.55	KW24D5 SIM HAL driver1841

Chapter System Mode Controller (SMC)

54.1	Overview1849
54.2	SMC HAL driver1850
54.2.1	Overview1850
54.2.2	Power Mode Configuration APIs1850
54.2.3	Power Mode Protection APIs1850
54.2.4	Data Structure Documentation1853
54.2.5	Enumeration Type Documentation1853
54.2.6	Function Documentation1855

Contents

Section Number	Title	Page Number
Chapter	Clock Manager (Clock)	
55.1	Overview	1859
55.1.1	Clock Manager1875
55.2	Data Structure Documentation	1879
55.2.1	struct oscr_config_t1879
55.2.2	struct osc_user_config_t1879
55.2.3	struct rtc_osc_user_config_t1880
55.2.4	struct mcg_config_t1881
55.2.5	struct clock_manager_user_config_t1882
55.2.6	struct clock_notify_struct_t1882
55.2.7	struct clock_manager_callback_user_config_t1883
55.2.8	struct clock_manager_state_t1883
55.2.9	struct sim_config_t1884
55.2.10	struct clock_name_config_t1884
55.3	Macro Definition Documentation	1884
55.3.1	CPU_LPO_CLK_HZ1884
55.4	Typedef Documentation	1885
55.4.1	clock_manager_callback_t1885
55.5	Enumeration Type Documentation	1885
55.5.1	clock_systick_src_t1885
55.5.2	clock_names_t1885
55.5.3	clock_manager_error_code_t1886
55.5.4	clock_manager_notify_t1886
55.5.5	clock_manager_callback_type_t1886
55.5.6	clock_manager_policy_t1886
55.6	Function Documentation	1886
55.6.1	CLOCK_SYS_GetFreq1886
55.6.2	CLOCK_SYS_GetCoreClockFreq1887
55.6.3	CLOCK_SYS_GetSystemClockFreq1887
55.6.4	CLOCK_SYS_GetBusClockFreq1887
55.6.5	CLOCK_SYS_GetFlashClockFreq1888
55.6.6	CLOCK_SYS_GetLpoClockFreq1888
55.6.7	CLOCK_SYS_SetSystickSrc1888
55.6.8	CLOCK_SYS_GetSystickFreq1888
55.6.9	CLOCK_SYS_Init1889
55.6.10	CLOCK_SYS_UpdateConfiguration1890
55.6.11	CLOCK_SYS_SetConfiguration1890
55.6.12	CLOCK_SYS_GetCurrentConfiguration1891
55.6.13	CLOCK_SYS_GetErrorCallback1891
55.6.14	CLOCK_SYS_SetMcgMode1891

Contents

Section Number	Title	Page Number
55.6.15	CLOCK_SYS_OscInit1892
55.6.16	CLOCK_SYS_OscDeinit1892
55.6.17	CLOCK_SYS_SetOscerConfiguration1892
55.6.18	CLOCK_SYS_SetOutDiv11893
55.6.19	CLOCK_SYS_GetOutDiv11893
55.6.20	CLOCK_SYS_SetOutDiv21893
55.6.21	CLOCK_SYS_GetOutDiv21893
55.6.22	CLOCK_SYS_SetOutDiv41894
55.6.23	CLOCK_SYS_GetOutDiv41895
55.6.24	CLOCK_SYS_SetOutDiv1895
55.6.25	CLOCK_SYS_GetOutDiv1895
55.6.26	CLOCK_SYS_GetPllFllClockFreq1896
55.6.27	CLOCK_SYS_SetPllfllSel1896
55.6.28	CLOCK_SYS_GetPllfllSel1896
55.6.29	CLOCK_SYS_GetFixedFreqClockFreq1896
55.6.30	CLOCK_SYS_GetInternalRefClockFreq1896
55.6.31	CLOCK_SYS_GetExternalRefClock32kFreq1897
55.6.32	CLOCK_SYS_SetExternalRefClock32kSrc1897
55.6.33	CLOCK_SYS_GetExternalRefClock32kSrc1897
55.6.34	CLOCK_SYS_GetOsc0ExternalRefClockFreq1897
55.6.35	CLOCK_SYS_GetOsc0ExternalRefClockUndivFreq1898
55.6.36	CLOCK_SYS_GetWdogFreq1898
55.6.37	CLOCK_SYS_GetTraceSrc1898
55.6.38	CLOCK_SYS_SetTraceSrc1898
55.6.39	CLOCK_SYS_GetTraceFreq1899
55.6.40	CLOCK_SYS_GetPortFilterFreq1899
55.6.41	CLOCK_SYS_GetLptmrFreq1900
55.6.42	CLOCK_SYS_GetEwmFreq1901
55.6.43	CLOCK_SYS_GetFtfFreq1901
55.6.44	CLOCK_SYS_GetCrcFreq1902
55.6.45	CLOCK_SYS_GetCmpFreq1903
55.6.46	CLOCK_SYS_GetVrefFreq1904
55.6.47	CLOCK_SYS_GetPdbFreq1904
55.6.48	CLOCK_SYS_GetPitFreq1905
55.6.49	CLOCK_SYS_GetSpiFreq1905
55.6.50	CLOCK_SYS_GetI2cFreq1906
55.6.51	CLOCK_SYS_GetAdcAltFreq1907
55.6.52	CLOCK_SYS_GetAdcAlt2Freq1907
55.6.53	CLOCK_SYS_GetFtmFixedFreq1908
55.6.54	CLOCK_SYS_GetFtmSystemClockFreq1909
55.6.55	CLOCK_SYS_GetFtmExternalFreq1909
55.6.56	CLOCK_SYS_GetFtmExternalSrc1909
55.6.57	CLOCK_SYS_SetFtmExternalSrc1910
55.6.58	CLOCK_SYS_GetUartFreq1910
55.6.59	CLOCK_SYS_GetGpioFreq1911

Contents

Section Number	Title	Page Number
55.6.60	CLOCK_SYS_EnableDmaClock1911
55.6.61	CLOCK_SYS_DisableDmaClock1912
55.6.62	CLOCK_SYS_GetDmaGateCmd1913
55.6.63	CLOCK_SYS_EnableDmamuxClock1913
55.6.64	CLOCK_SYS_DisableDmamuxClock1913
55.6.65	CLOCK_SYS_GetDmamuxGateCmd1913
55.6.66	CLOCK_SYS_EnablePortClock1914
55.6.67	CLOCK_SYS_DisablePortClock1914
55.6.68	CLOCK_SYS_GetPortGateCmd1914
55.6.69	CLOCK_SYS_EnableEwmClock1914
55.6.70	CLOCK_SYS_DisableEwmClock1915
55.6.71	CLOCK_SYS_GetEwmGateCmd1915
55.6.72	CLOCK_SYS_EnableFtfClock1915
55.6.73	CLOCK_SYS_DisableFtfClock1915
55.6.74	CLOCK_SYS_GetFtfGateCmd1916
55.6.75	CLOCK_SYS_EnableCrcClock1916
55.6.76	CLOCK_SYS_DisableCrcClock1916
55.6.77	CLOCK_SYS_GetCrcGateCmd1916
55.6.78	CLOCK_SYS_EnableAdcClock1917
55.6.79	CLOCK_SYS_DisableAdcClock1917
55.6.80	CLOCK_SYS_GetAdcGateCmd1917
55.6.81	CLOCK_SYS_EnableCmpClock1918
55.6.82	CLOCK_SYS_DisableCmpClock1918
55.6.83	CLOCK_SYS_GetCmpGateCmd1918
55.6.84	CLOCK_SYS_EnableDacClock1919
55.6.85	CLOCK_SYS_DisableDacClock1919
55.6.86	CLOCK_SYS_GetDacGateCmd1919
55.6.87	CLOCK_SYS_EnableVrefClock1920
55.6.88	CLOCK_SYS_DisableVrefClock1920
55.6.89	CLOCK_SYS_GetVrefGateCmd1920
55.6.90	CLOCK_SYS_EnablePdbClock1921
55.6.91	CLOCK_SYS_DisablePdbClock1921
55.6.92	CLOCK_SYS_GetPdbGateCmd1921
55.6.93	CLOCK_SYS_EnableFtmClock1922
55.6.94	CLOCK_SYS_DisableFtmClock1923
55.6.95	CLOCK_SYS_GetFtmGateCmd1923
55.6.96	CLOCK_SYS_EnablePitClock1924
55.6.97	CLOCK_SYS_DisablePitClock1925
55.6.98	CLOCK_SYS_GetPitGateCmd1925
55.6.99	CLOCK_SYS_EnableLptmrClock1925
55.6.100	CLOCK_SYS_DisableLptmrClock1926
55.6.101	CLOCK_SYS_GetLptmrGateCmd1926
55.6.102	CLOCK_SYS_EnableSpiClock1927
55.6.103	CLOCK_SYS_DisableSpiClock1927
55.6.104	CLOCK_SYS_GetSpiGateCmd1927

Contents

Section Number	Title	Page Number
55.6.105	CLOCK_SYS_EnableI2cClock1928
55.6.106	CLOCK_SYS_DisableI2cClock1928
55.6.107	CLOCK_SYS_GetI2cGateCmd1929
55.6.108	CLOCK_SYS_EnableUartClock1929
55.6.109	CLOCK_SYS_DisableUartClock1929
55.6.110	CLOCK_SYS_GetUartGateCmd1930
55.6.111	CLOCK_SYS_SetFtmExternalFreq1930
55.6.112	CLOCK_SYS_SetOutDiv31931
55.6.113	CLOCK_SYS_GetOutDiv31932
55.6.114	CLOCK_SYS_GetFlexbusFreq1932
55.6.115	CLOCK_SYS_GetRtcFreq1932
55.6.116	CLOCK_SYS_GetRtcOutFreq1932
55.6.117	CLOCK_SYS_GetRtcOutSrc1933
55.6.118	CLOCK_SYS_SetRtcOutSrc1933
55.6.119	CLOCK_SYS_GetEnetRmiiSrc1933
55.6.120	CLOCK_SYS_SetEnetRmiiSrc1933
55.6.121	CLOCK_SYS_GetEnetRmiiFreq1934
55.6.122	CLOCK_SYS_GetFlexcanFreq1934
55.6.123	CLOCK_SYS_SetEnetTimeStampSrc1934
55.6.124	CLOCK_SYS_GetEnetTimeStampSrc1935
55.6.125	CLOCK_SYS_GetEnetTimeStampFreq1935
55.6.126	CLOCK_SYS_GetCmtFreq1935
55.6.127	CLOCK_SYS_GetSdhcFreq1936
55.6.128	CLOCK_SYS_SetSdhcSrc1936
55.6.129	CLOCK_SYS_GetSdhcSrc1936
55.6.130	CLOCK_SYS_GetSaiFreq1937
55.6.131	CLOCK_SYS_GetUsbdcdFreq1937
55.6.132	CLOCK_SYS_EnableMpuClock1937
55.6.133	CLOCK_SYS_DisableMpuClock1938
55.6.134	CLOCK_SYS_GetMpuGateCmd1938
55.6.135	CLOCK_SYS_EnableFlexbusClock1938
55.6.136	CLOCK_SYS_DisableFlexbusClock1938
55.6.137	CLOCK_SYS_GetFlexbusGateCmd1939
55.6.138	CLOCK_SYS_EnableRngaClock1939
55.6.139	CLOCK_SYS_DisableRngaClock1939
55.6.140	CLOCK_SYS_GetRngaGateCmd1939
55.6.141	CLOCK_SYS_EnableSaiClock1940
55.6.142	CLOCK_SYS_DisableSaiClock1940
55.6.143	CLOCK_SYS_GetSaiGateCmd1940
55.6.144	CLOCK_SYS_EnableCmtClock1941
55.6.145	CLOCK_SYS_DisableCmtClock1942
55.6.146	CLOCK_SYS_GetCmtGateCmd1942
55.6.147	CLOCK_SYS_EnableRtcClock1942
55.6.148	CLOCK_SYS_DisableRtcClock1942
55.6.149	CLOCK_SYS_GetRtcGateCmd1943

Contents

Section Number	Title	Page Number
55.6.150	CLOCK_SYS_EnableEnetClock1943
55.6.151	CLOCK_SYS_DisableEnetClock1943
55.6.152	CLOCK_SYS_GetEnetGateCmd1943
55.6.153	CLOCK_SYS_EnableFlexcanClock1944
55.6.154	CLOCK_SYS_DisableFlexcanClock1944
55.6.155	CLOCK_SYS_GetFlexcanGateCmd1944
55.6.156	CLOCK_SYS_EnableSdhcClock1945
55.6.157	CLOCK_SYS_DisableSdhcClock1945
55.6.158	CLOCK_SYS_GetSdhcGateCmd1945
55.6.159	CLOCK_SYS_EnableTsiClock1946
55.6.160	CLOCK_SYS_DisableTsiClock1946
55.6.161	CLOCK_SYS_GetTsiGateCmd1946
55.6.162	CLOCK_SYS_EnableLlwuClock1947
55.6.163	CLOCK_SYS_DisableLlwuClock1948
55.6.164	CLOCK_SYS_GetLlwuGateCmd1948
55.6.165	CLOCK_SYS_SetEnetExternalFreq1948
55.6.166	CLOCK_SYS_SetSdhcExternalFreq1948
55.6.167	CLOCK_SYS_GetUsbfsFreq1949
55.6.168	CLOCK_SYS_SetUsbfsDiv1949
55.6.169	CLOCK_SYS_GetUsbfsDiv1950
55.6.170	CLOCK_SYS_EnableUsbfsClock1951
55.6.171	CLOCK_SYS_DisableUsbfsClock1951
55.6.172	CLOCK_SYS_GetUsbfsGateCmd1951
55.6.173	CLOCK_SYS_EnableUsbdcdClock1952
55.6.174	CLOCK_SYS_DisableUsbdcdClock1952
55.6.175	CLOCK_SYS_GetUsbdcdGateCmd1952
55.6.176	CLOCK_SYS_GetUsbfsSrc1953
55.6.177	CLOCK_SYS_SetUsbfsSrc1953
55.6.178	CLOCK_SYS_Osc0Deinit1954
55.6.179	CLOCK_SYS_SetUsbExternalFreq1954
55.6.180	CLOCK_SYS_GetLpuartSrc1954
55.6.181	CLOCK_SYS_SetLpuartSrc1955
55.6.182	CLOCK_SYS_GetLpuartFreq1956
55.6.183	CLOCK_SYS_EnableLpuartClock1956
55.6.184	CLOCK_SYS_DisableLpuartClock1957
55.6.185	CLOCK_SYS_GetLpuartGateCmd1957
55.6.186	CLOCK_SYS_GetPllFIIDivClockFreq1957
55.6.187	CLOCK_SYS_GetTpmFreq1958
55.6.188	CLOCK_SYS_SetTpmSrc1958
55.6.189	CLOCK_SYS_GetTpmSrc1959
55.6.190	CLOCK_SYS_GetTpmExternalFreq1959
55.6.191	CLOCK_SYS_SetTpmExternalSrc1960
55.6.192	CLOCK_SYS_GetTpmExternalSrc1960
55.6.193	CLOCK_SYS_GetUsbhsSlowClockSrc1960
55.6.194	CLOCK_SYS_SetUsbhsSlowClockSrc1961

Contents

Section Number	Title	Page Number
55.6.195	CLOCK_SYS_GetUsbhsSlowClockFreq1961
55.6.196	CLOCK_SYS_EnableSdramcClock1961
55.6.197	CLOCK_SYS_DisableSdramcClock1962
55.6.198	CLOCK_SYS_GetSdramcGateCmd1962
55.6.199	CLOCK_SYS_EnableTpmClock1962
55.6.200	CLOCK_SYS_DisableTpmClock1962
55.6.201	CLOCK_SYS_GetTpmGateCmd1963
55.6.202	CLOCK_SYS_EnableUsbhsClock1963
55.6.203	CLOCK_SYS_DisableUsbhsClock1963
55.6.204	CLOCK_SYS_GetUsbhsGateCmd1963
55.6.205	CLOCK_SYS_EnableUsbphyClock1964
55.6.206	CLOCK_SYS_DisableUsbphyClock1964
55.6.207	CLOCK_SYS_GetUsbphyGateCmd1964
55.6.208	CLOCK_SYS_EnableUsbhsdcdClock1965
55.6.209	CLOCK_SYS_DisableUsbhsdcdClock1966
55.6.210	CLOCK_SYS_GetUsbhsdcdGateCmd1966
55.6.211	CLOCK_SYS_GetFllClockFreq1966
55.6.212	CLOCK_SYS_GetCopFreq1966
55.6.213	CLOCK_SYS_GetLpisciFreq1967
55.6.214	CLOCK_SYS_SetLpisciSrc1967
55.6.215	CLOCK_SYS_GetLpisciSrc1967
55.6.216	CLOCK_SYS_EnableLpisciClock1968
55.6.217	CLOCK_SYS_DisableLpisciClock1968
55.6.218	CLOCK_SYS_GetLpisciGateCmd1968
55.6.219	CLOCK_SYS_SetTpmExternalFreq1968
55.6.220	CLOCK_SYS_GetCopFreq1969
55.6.221	CLOCK_SYS_SetCopSrc1969
55.6.222	CLOCK_SYS_GetCopSrc1969
55.6.223	CLOCK_SYS_GetFlexioFreq1969
55.6.224	CLOCK_SYS_SetFlexioSrc1970
55.6.225	CLOCK_SYS_GetFlexioSrc1970
55.6.226	CLOCK_SYS_EnableFlexioClock1971
55.6.227	CLOCK_SYS_DisableFlexioClock1971
55.6.228	CLOCK_SYS_GetFlexioGateCmd1971
55.6.229	CLOCK_SYS_EnableXrdcClock1972
55.6.230	CLOCK_SYS_DisableXrdcClock1972
55.6.231	CLOCK_SYS_GetXrdcGateCmd1972
55.6.232	CLOCK_SYS_EnableSemaClock1972
55.6.233	CLOCK_SYS_DisableSemaClock1973
55.6.234	CLOCK_SYS_GetSemaGateCmd1973
55.6.235	CLOCK_SYS_EnableFlashClock1973
55.6.236	CLOCK_SYS_DisableFlashClock1973
55.6.237	CLOCK_SYS_GetFlashGateCmd1974
55.6.238	CLOCK_SYS_EnableMuClock1974
55.6.239	CLOCK_SYS_DisableMuClock1974

Contents

Section Number	Title	Page Number
55.6.240	CLOCK_SYS_GetMuGateCmd1974
55.6.241	CLOCK_SYS_EnableIntmuxClock1975
55.6.242	CLOCK_SYS_DisableIntmuxClock1975
55.6.243	CLOCK_SYS_GetIntmuxGateCmd1975
55.6.244	CLOCK_SYS_GetPitSrc1975
55.6.245	CLOCK_SYS_SetPitSrc1976
55.6.246	CLOCK_SYS_EnableLpspiClock1976
55.6.247	CLOCK_SYS_DisableLpspiClock1976
55.6.248	CLOCK_SYS_GetLpspiGateCmd1976
55.6.249	CLOCK_SYS_GetLpspiSrc1977
55.6.250	CLOCK_SYS_SetLpspiSrc1977
55.6.251	CLOCK_SYS_GetLpspiFreq1977
55.6.252	CLOCK_SYS_EnableLpi2cClock1978
55.6.253	CLOCK_SYS_DisableLpi2cClock1979
55.6.254	CLOCK_SYS_GetLpi2cGateCmd1979
55.6.255	CLOCK_SYS_GetLpi2cSrc1979
55.6.256	CLOCK_SYS_SetLpi2cSrc1979
55.6.257	CLOCK_SYS_GetLpi2cFreq1980
55.6.258	CLOCK_SYS_EnableEvmsimClock1980
55.6.259	CLOCK_SYS_DisableEvmsimClock1980
55.6.260	CLOCK_SYS_GetEvmsimGateCmd1980
55.6.261	CLOCK_SYS_GetEvmsimSrc1981
55.6.262	CLOCK_SYS_SetEvmsimSrc1981
55.6.263	CLOCK_SYS_GetEvmsimFreq1981
55.6.264	CLOCK_SYS_EnableAtxClock1982
55.6.265	CLOCK_SYS_DisableAtxClock1982
55.6.266	CLOCK_SYS_GetAtxGateCmd1982
55.6.267	CLOCK_SYS_EnableTrngClock1982
55.6.268	CLOCK_SYS_DisableTrngClock1983
55.6.269	CLOCK_SYS_GetTrngGateCmd1983
55.6.270	CLOCK_SYS_SetOutDiv51983
55.6.271	CLOCK_SYS_GetOutDiv51983
55.6.272	CLOCK_SYS_GetOutdiv5ClockFreq1984
55.6.273	CLOCK_SYS_GetFastPeripheralClockFreq1984
55.6.274	CLOCK_SYS_GetDmaFreq1984
55.6.275	CLOCK_SYS_GetDmamuxFreq1984
55.6.276	CLOCK_SYS_GetAdcFreq1984
55.6.277	CLOCK_SYS_GetPwmFreq1985
55.6.278	CLOCK_SYS_GetEncFreq1985
55.6.279	CLOCK_SYS_GetXbarFreq1985
55.6.280	CLOCK_SYS_GetAoiFreq1986
55.6.281	CLOCK_SYS_GetFlexcanFreq1986
55.6.282	CLOCK_SYS_EnablePwmClock1986
55.6.283	CLOCK_SYS_DisablePwmClock1987
55.6.284	CLOCK_SYS_GetPwmGateCmd1987

Contents

Section Number	Title	Page Number
55.6.285	CLOCK_SYS_EnableAoiClock1987
55.6.286	CLOCK_SYS_DisableAoiClock1987
55.6.287	CLOCK_SYS_GetAoiGateCmd1988
55.6.288	CLOCK_SYS_EnableXbarClock1988
55.6.289	CLOCK_SYS_DisableXbarClock1988
55.6.290	CLOCK_SYS_GetXbarGateCmd1988
55.6.291	CLOCK_SYS_EnableEncClock1989
55.6.292	CLOCK_SYS_DisableEncClock1989
55.6.293	CLOCK_SYS_GetEncGateCmd1989
55.7	Variable Documentation1990
55.7.1	g_simBase1990
55.7.2	g_mcgBase1990
55.7.3	g_oscBase1990
55.8	K02F128101991
55.8.1	Overview1991
55.8.2	Data Structure Documentation1991
55.8.3	Macro Definition Documentation1992
55.8.4	Variable Documentation1992
55.9	K10D101993
55.9.1	Overview1993
55.9.2	Data Structure Documentation1993
55.9.3	Macro Definition Documentation1994
55.9.4	Variable Documentation1994
55.10	K11DA51995
55.10.1	Overview1995
55.10.2	Data Structure Documentation1995
55.10.3	Macro Definition Documentation1996
55.10.4	Variable Documentation1996
55.11	K20D101997
55.11.1	Overview1997
55.11.2	Data Structure Documentation1997
55.11.3	Macro Definition Documentation1998
55.11.4	Variable Documentation1998
55.12	K21DA51999
55.12.1	Overview1999
55.12.2	Data Structure Documentation1999
55.12.3	Macro Definition Documentation2000
55.12.4	Variable Documentation2000
55.13	K21FA122001

Contents

Section Number	Title	Page Number
55.13.1	Overview	2001
55.13.2	Data Structure Documentation	2001
55.13.3	Macro Definition Documentation	2002
55.13.4	Variable Documentation	2002
55.14	K22F12810	2003
55.14.1	Overview	2003
55.14.2	Data Structure Documentation	2003
55.14.3	Macro Definition Documentation	2004
55.14.4	Variable Documentation	2004
55.15	K22F25612	2005
55.15.1	Overview	2005
55.15.2	Data Structure Documentation	2005
55.15.3	Macro Definition Documentation	2006
55.15.4	Variable Documentation	2006
55.16	K22F51212	2007
55.16.1	Overview	2007
55.16.2	Data Structure Documentation	2007
55.16.3	Macro Definition Documentation	2008
55.16.4	Variable Documentation	2008
55.17	K24F12	2009
55.17.1	Overview	2009
55.17.2	Data Structure Documentation	2009
55.17.3	Macro Definition Documentation	2010
55.17.4	Variable Documentation	2010
55.18	K24F25612	2011
55.18.1	Overview	2011
55.18.2	Data Structure Documentation	2011
55.18.3	Macro Definition Documentation	2012
55.18.4	Variable Documentation	2012
55.19	K30D10	2013
55.19.1	Overview	2013
55.19.2	Data Structure Documentation	2013
55.19.3	Macro Definition Documentation	2014
55.19.4	Variable Documentation	2014
55.20	K40D10	2015
55.20.1	Overview	2015
55.20.2	Data Structure Documentation	2015
55.20.3	Macro Definition Documentation	2016
55.20.4	Variable Documentation	2016

Contents

Section Number	Title	Page Number
55.21	K50D10	2017
55.21.1	Overview	2017
55.21.2	Data Structure Documentation	2017
55.21.3	Macro Definition Documentation	2018
55.21.4	Variable Documentation	2018
55.22	K51D10	2019
55.22.1	Overview	2019
55.22.2	Data Structure Documentation	2019
55.22.3	Macro Definition Documentation	2020
55.22.4	Variable Documentation	2020
55.23	K52D10	2021
55.23.1	Overview	2021
55.23.2	Data Structure Documentation	2021
55.23.3	Macro Definition Documentation	2022
55.23.4	Variable Documentation	2022
55.24	K53D10	2023
55.24.1	Overview	2023
55.24.2	Data Structure Documentation	2023
55.24.3	Macro Definition Documentation	2024
55.24.4	Variable Documentation	2024
55.25	K60D10	2025
55.25.1	Overview	2025
55.25.2	Data Structure Documentation	2026
55.25.3	Macro Definition Documentation	2026
55.25.4	Variable Documentation	2026
55.26	K63F12	2027
55.26.1	Overview	2027
55.26.2	Data Structure Documentation	2027
55.26.3	Macro Definition Documentation	2028
55.26.4	Variable Documentation	2028
55.27	K64F12	2029
55.27.1	Overview	2029
55.27.2	Data Structure Documentation	2031
55.27.3	Macro Definition Documentation	2034
55.27.4	Variable Documentation	2034
55.28	KL02Z4	2035
55.28.1	Overview	2035
55.28.2	Data Structure Documentation	2035
55.28.3	Macro Definition Documentation	2036

Contents

Section Number	Title	Page Number
55.28.4	Variable Documentation	2036
55.29	KL03Z4	2037
55.29.1	Overview	2037
55.29.2	Data Structure Documentation	2037
55.29.3	Macro Definition Documentation	2038
55.29.4	Variable Documentation	2038
55.30	KL13Z644	2039
55.31	KL14Z4	2040
55.31.1	Overview	2040
55.31.2	Data Structure Documentation	2040
55.31.3	Macro Definition Documentation	2041
55.31.4	Variable Documentation	2041
55.32	KL15Z4	2042
55.32.1	Overview	2042
55.32.2	Data Structure Documentation	2042
55.32.3	Macro Definition Documentation	2043
55.32.4	Variable Documentation	2043
55.33	KL16Z4	2044
55.33.1	Overview	2044
55.33.2	Data Structure Documentation	2044
55.33.3	Macro Definition Documentation	2045
55.33.4	Variable Documentation	2045
55.34	KL17Z4	2046
55.34.1	Overview	2046
55.34.2	Data Structure Documentation	2046
55.34.3	Macro Definition Documentation	2047
55.34.4	Variable Documentation	2047
55.35	KL17Z644	2048
55.35.1	Overview	2048
55.35.2	Data Structure Documentation	2048
55.35.3	Macro Definition Documentation	2049
55.35.4	Variable Documentation	2049
55.36	KL24Z4	2050
55.36.1	Overview	2050
55.36.2	Data Structure Documentation	2050
55.36.3	Macro Definition Documentation	2051
55.36.4	Variable Documentation	2051

Contents

Section Number	Title	Page Number
55.37	KL25Z4	2052
55.37.1	Overview	2052
55.37.2	Data Structure Documentation	2052
55.37.3	Macro Definition Documentation	2053
55.37.4	Variable Documentation	2053
55.38	KL26Z4	2054
55.38.1	Overview	2054
55.38.2	Data Structure Documentation	2054
55.38.3	Macro Definition Documentation	2055
55.38.4	Variable Documentation	2055
55.39	KL27Z4	2056
55.39.1	Overview	2056
55.39.2	Data Structure Documentation	2056
55.39.3	Macro Definition Documentation	2057
55.39.4	Variable Documentation	2057
55.40	KL127Z644	2058
55.40.1	Overview	2058
55.40.2	Data Structure Documentation	2058
55.40.3	Macro Definition Documentation	2059
55.40.4	Variable Documentation	2059
55.41	K128T7	2060
55.41.1	Overview	2060
55.41.2	Data Structure Documentation	2060
55.41.3	Macro Definition Documentation	2061
55.41.4	Variable Documentation	2061
55.42	KL33Z4	2062
55.42.1	Overview	2062
55.42.2	Data Structure Documentation	2062
55.42.3	Macro Definition Documentation	2063
55.42.4	Variable Documentation	2063
55.43	KL33Z644	2064
55.44	KL34Z4	2065
55.44.1	Overview	2065
55.44.2	Data Structure Documentation	2065
55.44.3	Macro Definition Documentation	2066
55.44.4	Variable Documentation	2066
55.45	KL36Z4	2067
55.45.1	Overview	2067

Contents

Section Number	Title	Page Number
55.45.2	Data Structure Documentation	2067
55.45.3	Macro Definition Documentation	2068
55.45.4	Variable Documentation	2068
55.46	KL43Z4	2069
55.46.1	Overview	2069
55.46.2	Data Structure Documentation	2069
55.46.3	Macro Definition Documentation	2070
55.46.4	Variable Documentation	2070
55.47	KL46Z4	2071
55.47.1	Overview	2071
55.47.2	Data Structure Documentation	2072
55.47.3	Macro Definition Documentation	2073
55.47.4	Variable Documentation	2073
55.48	KV10Z7	2074
55.48.1	Overview	2074
55.48.2	Data Structure Documentation	2074
55.48.3	Macro Definition Documentation	2075
55.48.4	Variable Documentation	2075
55.49	KV30F12810	2076
55.49.1	Overview	2076
55.49.2	Data Structure Documentation	2076
55.49.3	Macro Definition Documentation	2077
55.49.4	Variable Documentation	2077
55.50	KV31F12810	2078
55.50.1	Overview	2078
55.50.2	Data Structure Documentation	2078
55.50.3	Macro Definition Documentation	2079
55.50.4	Variable Documentation	2079
55.51	KV31F25612	2080
55.51.1	Overview	2080
55.51.2	Data Structure Documentation	2080
55.51.3	Macro Definition Documentation	2081
55.51.4	Variable Documentation	2081
55.52	KV31F51212	2082
55.52.1	Overview	2082
55.52.2	Data Structure Documentation	2082
55.52.3	Macro Definition Documentation	2083
55.52.4	Variable Documentation	2083

Contents

Section Number	Title	Page Number
55.53	KW21D5	2084
55.53.1	Overview	2084
55.53.2	Data Structure Documentation	2084
55.53.3	Macro Definition Documentation	2085
55.53.4	Variable Documentation	2085
55.54	KW22D5	2086
55.54.1	Overview	2086
55.54.2	Data Structure Documentation	2086
55.54.3	Macro Definition Documentation	2087
55.54.4	Variable Documentation	2087
55.55	KW24D5	2088
55.55.1	Overview	2088
55.55.2	Data Structure Documentation	2088
55.55.3	Macro Definition Documentation	2089
55.55.4	Variable Documentation	2089
Chapter	Hwtimer_driver	
56.1	Overview	2091
56.2	Data Structure Documentation	2093
56.2.1	struct hwtimer_t	2093
56.2.2	struct hwtimer_time_t	2094
56.2.3	struct hwtimer_devif_t	2094
56.3	Enumeration Type Documentation	2095
56.3.1	_hwtimer_error_code_t	2095
56.4	Function Documentation	2095
56.4.1	HWTIMER_SYS_Init	2095
56.4.2	HWTIMER_SYS_Deinit	2096
56.4.3	HWTIMER_SYS_SetPeriod	2097
56.4.4	HWTIMER_SYS_GetPeriod	2098
56.4.5	HWTIMER_SYS_Start	2099
56.4.6	HWTIMER_SYS_Stop	2100
56.4.7	HWTIMER_SYS_GetModulo	2101
56.4.8	HWTIMER_SYS_GetTime	2102
56.4.9	HWTIMER_SYS_GetTicks	2102
56.4.10	HWTIMER_SYS_RegisterCallback	2103
56.4.11	HWTIMER_SYS_BlockCallback	2104
56.4.12	HWTIMER_SYS_UnblockCallback	2105
56.4.13	HWTIMER_SYS_CancelCallback	2106
56.5	HwTimer Driver	2107

Contents

Section Number	Title	Page Number
56.5.1	HwTimer Overview	2107
56.6	HwTimer Interrupt handlers	2108
56.6.1	HwTimer Interrupt handler overview	2108
 Chapter Interrupt Manager (Interrupt)		
57.1	Overview	2109
57.1.1	Interrupt Manager	2109
57.2	Enumeration Type Documentation	2110
57.2.1	interrupt_status_t	2110
57.3	Function Documentation	2110
57.3.1	INT_SYS_InstallHandler	2110
57.3.2	INT_SYS_EnableIRQ	2110
57.3.3	INT_SYS_DisableIRQ	2111
57.3.4	INT_SYS_EnableIRQGlobal	2111
57.3.5	INT_SYS_DisableIRQGlobal	2111
 Chapter Power Manager (Power)		
58.1	Overview	2113
58.2	Data Structure Documentation	2115
58.2.1	struct power_manager_user_config_t	2115
58.2.2	struct power_manager_notify_struct_t	2116
58.2.3	struct power_manager_callback_user_config_t	2116
58.2.4	struct power_manager_state_t	2116
58.3	Typedef Documentation	2117
58.3.1	power_manager_callback_data_t	2117
58.3.2	power_manager_callback_t	2117
58.4	Enumeration Type Documentation	2118
58.4.1	power_manager_modes_t	2118
58.4.2	power_manager_error_code_t	2119
58.4.3	power_manager_policy_t	2119
58.4.4	power_manager_notify_t	2119
58.4.5	power_manager_callback_type_t	2120
58.5	Function Documentation	2120
58.5.1	POWER_SYS_Init	2120
58.5.2	POWER_SYS_Deinit	2121
58.5.3	POWER_SYS_SetMode	2122
58.5.4	POWER_SYS_GetLastMode	2123

Contents

Section Number	Title	Page Number
58.5.5	POWER_SYS_GetLastModeConfig	2123
58.5.6	POWER_SYS_GetCurrentMode	2123
58.5.7	POWER_SYS_GetErrorCallbackIndex	2124
58.5.8	POWER_SYS_GetErrorCallback	2124
58.5.9	POWER_SYS_GetVeryLowPowerModeStatus	2124
58.5.10	POWER_SYS_GetLowLeakageWakeupResetStatus	2124
58.5.11	POWER_SYS_GetAckIsolation	2125
58.5.12	POWER_SYS_ClearAckIsolation	2125
58.6	Power Manager driver	2126
58.6.1	Power Manager Overview	2126
Chapter	Utilities for the Kinetis SDK	
59.1	Debug Console Initialization	2129
Chapter	OS Abstraction Layer (OSA)	
60.1	Overview	2131
60.1.1	OS Abstraction Layer	2134
60.2	Typedef Documentation	2139
60.2.1	osa_int_handler_t	2139
60.3	Enumeration Type Documentation	2139
60.3.1	osa_status_t	2139
60.3.2	osa_event_clear_mode_t	2139
60.3.3	osa_critical_section_mode_t	2140
60.4	Function Documentation	2140
60.4.1	OSA_SemaCreate	2140
60.4.2	OSA_SemaWait	2140
60.4.3	OSA_SemaPost	2141
60.4.4	OSA_SemaDestroy	2142
60.4.5	OSA_MutexCreate	2143
60.4.6	OSA_MutexLock	2143
60.4.7	OSA_MutexUnlock	2144
60.4.8	OSA_MutexDestroy	2145
60.4.9	OSA_EventCreate	2146
60.4.10	OSA_EventWait	2146
60.4.11	OSA_EventSet	2147
60.4.12	OSA_EventClear	2148
60.4.13	OSA_EventGetFlags	2148
60.4.14	OSA_EventDestroy	2149
60.4.15	OSA_TaskCreate	2149

Contents

Section Number	Title	Page Number
60.4.16	OSA_TaskDestroy	2150
60.4.17	OSA_TaskYield	2151
60.4.18	OSA_TaskGetHandler	2151
60.4.19	OSA_TaskGetPriority	2151
60.4.20	OSA_TaskSetPriority	2152
60.4.21	OSA_MsgQCreate	2152
60.4.22	OSA_MsgQPut	2153
60.4.23	OSA_MsgQGet	2153
60.4.24	OSA_MsgQDestroy	2154
60.4.25	OSA_MemAlloc	2155
60.4.26	OSA_MemAllocZero	2155
60.4.27	OSA_MemFree	2155
60.4.28	OSA_TimeDelay	2155
60.4.29	OSA_TimeGetMsec	2156
60.4.30	OSA_InstallIntHandler	2156
60.4.31	OSA_EnterCritical	2156
60.4.32	OSA_ExitCritical	2156
60.4.33	OSA_Init	2157
60.4.34	OSA_Start	2157
60.5	Bare Metal Abstraction Layer	2158
60.5.1	Overview	2158
60.5.2	Data Structure Documentation	2162
60.5.3	Macro Definition Documentation	2165
60.5.4	Function Documentation	2166
60.6	MQX RTOS Abstraction Layer	2167
60.6.1	Overview	2167
60.6.2	Data Structure Documentation	2168
60.6.3	Macro Definition Documentation	2169
60.6.4	Typedef Documentation	2170
60.6.5	Function Documentation	2170
60.7	μC/OS-II Abstraction Layer	2171
60.7.1	Overview	2171
60.7.2	Data Structure Documentation	2172
60.7.3	Macro Definition Documentation	2173
60.8	μC/OS-III Abstraction Layer	2175
60.8.1	Overview	2175
60.8.2	Data Structure Documentation	2176
60.8.3	Macro Definition Documentation	2177
60.8.4	Typedef Documentation	2178
60.9	FreeRTOS Abstraction Layer	2179

Contents

Section Number	Title	Page Number
60.9.1	Overview	2179
60.9.2	Data Structure Documentation	2180
60.9.3	Macro Definition Documentation	2180
60.9.4	Typedef Documentation	2181
Chapter	Flexio_spi_driver	
61.1	Overview	2183
61.2	Data Structure Documentation	2185
61.2.1	struct flexio_spi_state_t	2185
61.2.2	struct flexio_spi_hwconfig_t	2186
61.2.3	struct flexio_spi_userconfig_t	2187
61.3	Enumeration Type Documentation	2187
61.3.1	flexio_spi_status_t	2187
61.3.2	flexio_spi_master_slave_mode_t	2187
61.3.3	flexio_spi_shift_direction_t	2188
61.3.4	flexio_spi_clock_phase_t	2188
61.3.5	flexio_spi_data_bitcount_mode_t	2188
61.4	Function Documentation	2188
61.4.1	FLEXIO_SPI_DRV_Init	2188
61.4.2	FLEXIO_SPI_DRV_Deinit	2189
61.4.3	FLEXIO_SPI_DRV_SendDataBlocking	2189
61.4.4	FLEXIO_SPI_DRV_SendData	2189
61.4.5	FLEXIO_SPI_DRV_GetTransmitStatus	2190
61.4.6	FLEXIO_SPI_DRV_AbortSendingData	2190
61.4.7	FLEXIO_SPI_DRV_ReceiveDataBlocking	2191
61.4.8	FLEXIO_SPI_DRV_ReceiveData	2191
61.4.9	FLEXIO_SPI_DRV_GetReceiveStatus	2192
61.4.10	FLEXIO_SPI_DRV_AbortReceivingData	2192
61.4.11	FLEXIO_SPI_DRV_TransferDataBlocking	2193
61.4.12	FLEXIO_SPI_DRV_TransferData	2193
61.4.13	FLEXIO_SPI_DRV_TX_IRQHandler	2194
61.4.14	FLEXIO_SPI_DRV_RX_IRQHandler	2194
61.4.15	FLEXIO_SPI_DRV_DmaSendDataBlocking	2194
61.4.16	FLEXIO_SPI_DRV_DmaSendData	2195
61.4.17	FLEXIO_SPI_DRV_DmaGetTransmitStatus	2196
61.4.18	FLEXIO_SPI_DRV_DmaAbortSendingData	2196
61.4.19	FLEXIO_SPI_DRV_DmaReceiveDataBlocking	2197
61.4.20	FLEXIO_SPI_DRV_DmaReceiveData	2197
61.4.21	FLEXIO_SPI_DRV_DmaGetReceiveStatus	2197
61.4.22	FLEXIO_SPI_DRV_AbortDmaReceivingData	2198
61.4.23	FLEXIO_SPI_DRV_DmaTransferDataBlocking	2198

Contents

Section Number	Title	Page Number
61.4.24	FLEXIO_SPI_DRV_DmaTransferData	2199
 Chapter Flexio_uart_driver		
62.1	Overview	2201
62.2	Data Structure Documentation	2202
62.2.1	struct flexio_uart_dmastate_t	2202
62.2.2	struct flexio_uartdma_userconfig_t	2203
62.2.3	struct flexio_uartdma_userconfig_t	2204
62.2.4	struct flexio_uart_edmastate_t	2204
62.3	Function Documentation	2205
62.3.1	FLEXIO_UART_DRV_DmaInit	2205
62.3.2	FLEXIO_UART_DRV_DmaDeinit	2205
62.3.3	FLEXIO_UART_DRV_DmaSendDataBlocking	2206
62.3.4	FLEXIO_UART_DRV_DmaSendData	2206
62.3.5	FLEXIO_UART_DRV_DmaGetTransmitStatus	2206
62.3.6	FLEXIO_UART_DRV_DmaAbortSendingData	2207
62.3.7	FLEXIO_UART_DRV_DmaReceiveDataBlocking	2207
62.3.8	FLEXIO_UART_DRV_DmaReceiveData	2208
62.3.9	FLEXIO_UART_DRV_DmaGetReceiveStatus	2208
62.3.10	FLEXIO_UART_DRV_DmaAbortReceivingData	2209
62.3.11	FLEXIO_UART_DRV_EdmaInit	2209
62.3.12	FLEXIO_UART_DRV_EdmaDeinit	2210
62.3.13	FLEXIO_UART_DRV_EdmaSendDataBlocking	2210
62.3.14	FLEXIO_UART_DRV_EdmaSendData	2211
62.3.15	FLEXIO_UART_DRV_EdmaGetTransmitStatus	2212
62.3.16	FLEXIO_UART_DRV_EdmaAbortSendingData	2212
62.3.17	FLEXIO_UART_DRV_EdmaReceiveDataBlocking	2213
62.3.18	FLEXIO_UART_DRV_EdmaReceiveData	2213
62.3.19	FLEXIO_UART_DRV_EdmaGetReceiveStatus	2213
62.3.20	FLEXIO_UART_DRV_EdmaAbortReceivingData	2214
 Chapter Flexio_spi_hal		
63.1	Overview	2215
63.2	Data Structure Documentation	2216
63.2.1	struct flexio_spi_dev_t	2216
63.2.2	struct flexio_spi_master_config_t	2217
63.2.3	struct flexio_spi_slave_config_t	2217
63.3	Function Documentation	2218
63.3.1	FLEXIO_SPI_HAL_ConfigMaster	2218

Contents

Section Number	Title	Page Number
63.3.2	FLEXIO_SPI_HAL_ConfigSlave	2218
63.3.3	FLEXIO_SPI_HAL_GetTxBufferEmptyFlag	2218
63.3.4	FLEXIO_SPI_HAL_ClearTxBufferEmptyFlag	2219
63.3.5	FLEXIO_SPI_HAL_SetTxBufferEmptyIntCmd	2219
63.3.6	FLEXIO_SPI_HAL_GetTxErrFlag	2219
63.3.7	FLEXIO_SPI_HAL_ClearTxErrFlag	2219
63.3.8	FLEXIO_SPI_HAL_SetTxErrIntCmd	2220
63.3.9	FLEXIO_SPI_HAL_PutDataMSB	2220
63.3.10	FLEXIO_SPI_HAL_PutDataMSBPolling	2220
63.3.11	FLEXIO_SPI_HAL_PutDataLSB	2220
63.3.12	FLEXIO_SPI_HAL_PutDataLSBPolling	2221
63.3.13	FLEXIO_SPI_HAL_SetTxDmaCmd	2221
63.3.14	FLEXIO_SPI_HAL_GetTxBufferMSBAddr	2221
63.3.15	FLEXIO_SPI_HAL_GetTxBufferLSBAddr	2221
63.3.16	FLEXIO_SPI_HAL_GetRxBufferFullFlag	2222
63.3.17	FLEXIO_SPI_HAL_ClearRxBufferFullFlag	2222
63.3.18	FLEXIO_SPI_HAL_SetRxBufferFullIntCmd	2222
63.3.19	FLEXIO_SPI_HAL_GetRxErrFlag	2222
63.3.20	FLEXIO_SPI_HAL_ClearRxErrFlag	2223
63.3.21	FLEXIO_SPI_HAL_SetRxErrIntCmd	2223
63.3.22	FLEXIO_SPI_HAL_GetDataMSB	2223
63.3.23	FLEXIO_SPI_HAL_GetDataMSBPolling	2223
63.3.24	FLEXIO_SPI_HAL_GetDataLSB	2224
63.3.25	FLEXIO_SPI_HAL_GetDataLSBPolling	2224
63.3.26	FLEXIO_SPI_HAL_SetRxDmaCmd	2224
63.3.27	FLEXIO_SPI_HAL_GetRxBufferMSBAddr	2225
63.3.28	FLEXIO_SPI_HAL_GetRxBufferLSBAddr	2225

Chapter Pwm_hal

64.1	Overview	2227
64.2	Data Structure Documentation	2231
64.2.1	struct pwm_module_setup_t	2231
64.2.2	struct pwm_fault_setup_t	2231
64.2.3	struct pwm_capture_setup_t	2231
64.3	Enumeration Type Documentation	2232
64.3.1	pwm_module_t	2232
64.3.2	pwm_val_regs_t	2232
64.3.3	pwm_status_t	2232
64.3.4	pwm_clock_src_t	2232
64.3.5	pwm_clock_ps_t	2233
64.3.6	pwm_force_output_trigger_t	2233
64.3.7	pwm_init_src_t	2233

Contents

Section Number	Title	Page Number
64.3.8	pwm_load_frequency_t	2234
64.3.9	pwm_fault_input_t	2234
64.3.10	pwm_capture_edge_t	2234
64.3.11	pwm_force_signal_t	2235
64.3.12	pwm_chnl_pair_operation_t	2235
64.3.13	pwm_reg_reload_t	2235
64.3.14	pwm_fault_recovery_mode_t	2235
64.4	Function Documentation	2235
64.4.1	PWM_HAL_Init	2235
64.4.2	PWM_HAL_SetupPwmSubModule	2236
64.4.3	PWM_HAL_SetupFaults	2236
64.4.4	PWM_HAL_SetupCapture	2237
64.4.5	PWM_HAL_GetCaptureValReg	2237
64.4.6	PWM_HAL_SetValReg	2238
64.4.7	PWM_HAL_SetupForceSignal	2239
64.4.8	PWM_HAL_GetCounter	2239
64.4.9	PWM_HAL_SetCounterInitVal	2240
64.4.10	PWM_HAL_SetForceCmd	2241
64.4.11	PWM_HAL_SetOutputPolarityPwmBCmd	2241
64.4.12	PWM_HAL_SetOutputPolarityPwmACmd	2241
64.4.13	PWM_HAL_SetOutputPolarityPwmXCmd	2242
64.4.14	PWM_HAL_SetOutputTriggerCmd	2242
64.4.15	PWM_HAL_SetPwmAFaultInputCmd	2243
64.4.16	PWM_HAL_SetPwmBFaultInputCmd	2244
64.4.17	PWM_HAL_SetPwmXFaultInputCmd	2244
64.4.18	PWM_HAL_SetOutputPwmXCmd	2245
64.4.19	PWM_HAL_SetOutputPwmBCmd	2246
64.4.20	PWM_HAL_SetOutputPwmACmd	2246
64.4.21	PWM_HAL_SetSwCtrlOutCmd	2246
64.4.22	PWM_HAL_SetPwmRunCmd	2247
64.4.23	PWM_HAL_SetFaultIntCmd	2247
64.4.24	PWM_HAL_ClearFaultFlags	2247

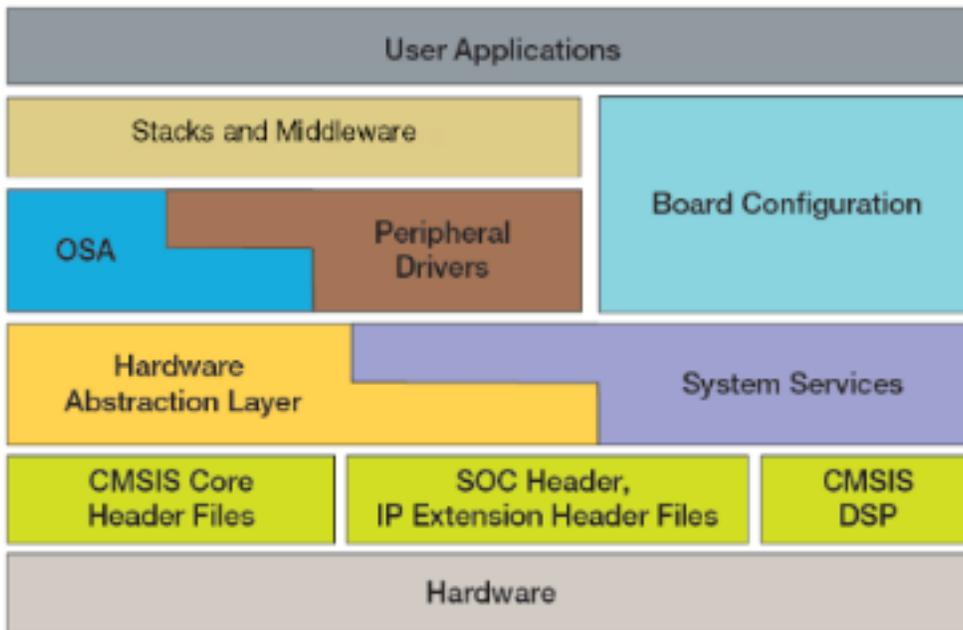
Chapter 1

Introduction

The Kinetis software development kit (KSDK) is an extensive suite of robust hardware interface and hardware abstraction layers, peripheral drivers, RTOS abstractions, stacks, and middleware designed to simplify and accelerate application development on Freescale Kinetis MCUs. The addition of Processor Expert technology for software and board configuration provides unmatched ease of use and flexibility.

Included in the Kinetis SDK is full source code under a permissive open-source license for all hardware abstraction and peripheral driver software. Mainline releases include support for a collection of Kinetis MCUs, whereas standalone releases offer support for one or a few additional Kinetis MCUs only. See the Release Notes and the Supported Devices section on www.freescale.com/ksdk for details.

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
- bare-metal (no RTOS)
- Stacks and middleware in source or object formats including:
 - a USB device, host, and OTG stack with comprehensive USB class support
 - CMSIS-DSP, a suite of common signal processing functions
 - 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 hardware abstraction layer, peripheral drivers, middleware, and RTOSes. All examples are provided with projects for the following toolchains:

- Atollic TrueSTUDIO
- GNU toolchain for ARM® Cortex®-M with Cmake build system
- IAR Embedded Workbench
- Keil MDK
- Kinetis Design Studio
- lwIP, a lightweight TCPIP networking stack

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 KSDK 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 and board configuration. Processor Expert is a complimentary PC-hosted software configuration tool (Eclipse plug-in) 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:

Deliverable	Location
Examples	<install_dir>/examples/...
Demo applications	<install_dir>/examples/<board_name>/demo_apps/...
Driver examples	<install_dir>/examples/<board_name>/driver_examples/...
Documentation	<install_dir>/doc/...
MQX Documentation	<install_dir>/doc/rtos/mqx/...
USB Documentation	<install_dir>/doc/usb/...
MQX RTCS Documentation	<install_dir>/doc/tcpip/mqx_rtcs/...
lwIP Documentation	<install_dir>/doc/tcpip/lwip/...
MQX MFS Documentation	<install_dir>/doc/filesystem/mqx_mfs/...
Projects to build libraries	<install_dir>/lib/...
Middleware	<install_dir>/middleware/...
TCP/IP stacks	<install_dir>/middleware/tcpip/...
File System	<install_dir>/middleware/filesystem/...
Driver library, startup code and utilities	<install_dir>/platform/...
Cortex Microcontroller Software Interface Standard (CMSIS) ARM Cortex®-M header files, DSP library source, and IP extension header files	<install_dir>/platform/CMSIS/...
Composite drivers for SD-card and Soundcard support	<install_dir>/platform/composite/...
Linker control files for each supported tool chain	<install_dir>/platform/devices/<board_name>/linker/...
CMSIS compliant Startup Code	<install_dir>/platform/devices/<board_name>/startup/...
Peripheral Drivers	<install_dir>/platform/drivers/...
Hardware Abstraction Layer	<install_dir>/platform/hal/...
OS Abstraction for Bare Metal and RTOS	<install_dir>/platform/osa/...
System Services such as clock manager, interrupt manager, unified hardware timer, and low power manager	<install_dir>/platform/system/...
Utilities such as debug console	<install_dir>/platform/utilities/...
RTOS Kernel Code, RTOS abstraction implementations, and RTOS kernel folders	<install_dir>/rtos/...
A Processor Expert service pack and cmake toolchain files.	<install_dir>/tools
USB stack	<install_dir>/usb/...
Utilities such as shell and standard libraries	<install_dir>/utilities/...

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

For the latest version of this and other Kinetis SDK documents, see the Kinetis SDK homepage www.freescale.com/ksdk.

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.

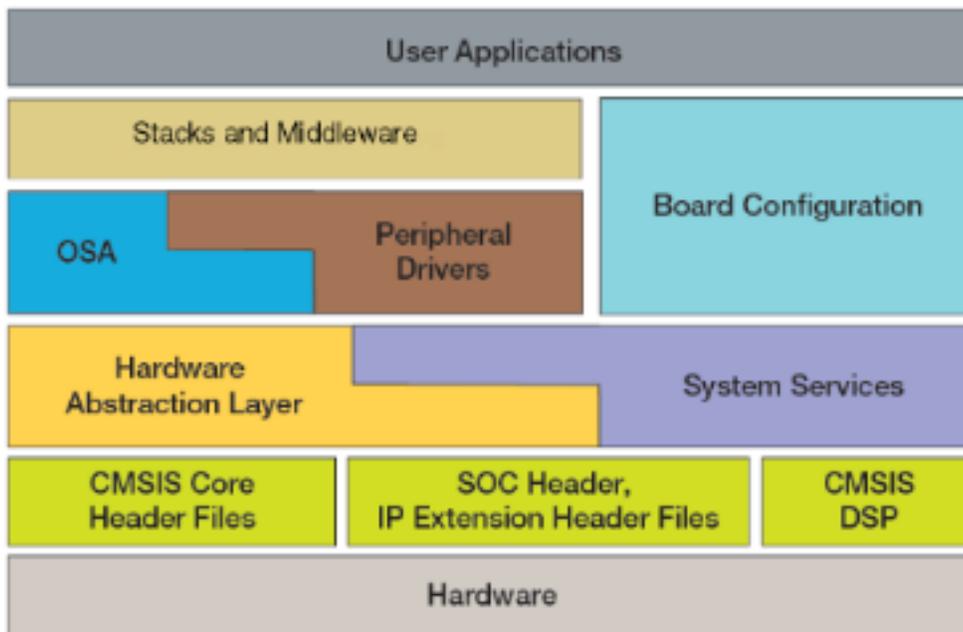
Overview

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, SOC Header, IP Extension Header Files, and CMSIS DSP libraries.
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. The extension header files take advantage of the bit band feature when reading from or writing to 1-bit wide fields within a register. Macros are provided in the SoC memory-mapped header files for accessing register bit fields. When accessing the register macros in the extension header file, the user must pass in the register base address for the desired peripheral as a pointer to `<IP>_Type` structure. 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 contain 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. The Clock Manager provides a notification framework which the software component of the KSDK, such as drivers, uses to register callback functions and execute the predefined code flow during the clock mode transition.

Power Manager

The Power Manager provides centralized power-related functions for the entire system. It dynamically sets the system power mode and configures the LLWU wakeup setting. The Clock Manager provides a notification framework which the software component of the KSDK, such as drivers, uses to register callback functions and execute the predefined code flow during the clock mode transition.

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. An overall Peripheral Feature Header File is provided for KSDK-supported MCU device to define the feature or configuration differences for each Kinetis sub-family device.

Design Guidelines

This section summarizes the design guidelines that were used 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 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. Note that HAL is used for MCUs with restricted RAM and FLASH size.

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.

Examples

The examples provided in the KSDK provide examples, which show how to build user applications using the KSDK framework. The examples can be found in the KSDK top-level demo directory. The KSDK

provides two classes of software examples

- **Demo Applications:** Full-featured applications intended to highlight key functionality of a MCU, focusing on a particular use case.
- **Driver Examples:** Simple applications intended to concisely illustrate how to use the KSDK's peripheral drivers.

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 `mcu-sdk/examples/<BOARD>`.

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 of 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 KSDK 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](#)

ADC16 HAL Driver

3.2 ADC16 HAL Driver

3.2.1 Overview

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

Data Structures

- struct `adc16_chn_config_t`
Defines the structure to configure the ADC16 channel. [More...](#)
- struct `adc16_converter_config_t`
Defines the structure to configure the ADC16's converter. [More...](#)
- struct `adc16_hw_cmp_config_t`
Defines the structure to configure the ADC16 internal comparator. [More...](#)

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_t` {
`kAdc16ClkDividerOf1` = 0U,
`kAdc16ClkDividerOf2` = 1U,
`kAdc16ClkDividerOf4` = 2U,
`kAdc16ClkDividerOf8` = 3U }
Defines the type of the enumerating divider for the converter.
- enum `adc16_resolution_t` {
`kAdc16ResolutionBitOf8or9` = 0U,
`kAdc16ResolutionBitOfSingleEndAs8` = `kAdc16ResolutionBitOf8or9`,
`kAdc16ResolutionBitOfDiffModeAs9` = `kAdc16ResolutionBitOf8or9`,
`kAdc16ResolutionBitOf12or13` = 1U,
`kAdc16ResolutionBitOfSingleEndAs12` = `kAdc16ResolutionBitOf12or13`,
`kAdc16ResolutionBitOfDiffModeAs13` = `kAdc16ResolutionBitOf12or13`,
`kAdc16ResolutionBitOf10or11` = 2U,
`kAdc16ResolutionBitOfSingleEndAs10` = `kAdc16ResolutionBitOf10or11`,
`kAdc16ResolutionBitOfDiffModeAs11` = `kAdc16ResolutionBitOf10or11` }
Defines the type of the enumerating resolution for the converter.
- enum `adc16_clk_src_mode_t` {
`kAdc16ClkSrcOfBusClk` = 0U,
`kAdc16ClkSrcOfAltClk2` = 1U,
`kAdc16ClkSrcOfAltClk` = 2U,
`kAdc16ClkSrcOfAsynClk` = 3U }
Defines the type of the enumerating source of the input clock.

- enum `adc16_long_sample_cycle_t` {
 `kAdc16LongSampleCycleOf24` = 0U,
 `kAdc16LongSampleCycleOf16` = 1U,
 `kAdc16LongSampleCycleOf10` = 2U,
 `kAdc16LongSampleCycleOf4` = 3U }
 Defines the type of the enumerating long sample cycles.
- enum `adc16_ref_volt_src_t` {
 `kAdc16RefVoltSrcOfVref` = 0U,
 `kAdc16RefVoltSrcOfValt` = 1U }
 Defines the type of the enumerating reference voltage source.
- enum `adc16_chn_t` {

ADC16 HAL Driver

```

kAdc16Chn0 = 0U,
kAdc16Chn1 = 1U,
kAdc16Chn2 = 2U,
kAdc16Chn3 = 3U,
kAdc16Chn4 = 4U,
kAdc16Chn5 = 5U,
kAdc16Chn6 = 6U,
kAdc16Chn7 = 7U,
kAdc16Chn8 = 8U,
kAdc16Chn9 = 9U,
kAdc16Chn10 = 10U,
kAdc16Chn11 = 11U,
kAdc16Chn12 = 12U,
kAdc16Chn13 = 13U,
kAdc16Chn14 = 14U,
kAdc16Chn15 = 15U,
kAdc16Chn16 = 16U,
kAdc16Chn17 = 17U,
kAdc16Chn18 = 18U,
kAdc16Chn19 = 19U,
kAdc16Chn20 = 20U,
kAdc16Chn21 = 21U,
kAdc16Chn22 = 22U,
kAdc16Chn23 = 23U,
kAdc16Chn24 = 24U,
kAdc16Chn25 = 25U,
kAdc16Chn26 = 26U,
kAdc16Chn27 = 27U,
kAdc16Chn28 = 28U,
kAdc16Chn29 = 29U,
kAdc16Chn30 = 30U,
kAdc16Chn31 = 31U,
kAdc16Chn0d = kAdc16Chn0,
kAdc16Chn1d = kAdc16Chn1,
kAdc16Chn2d = kAdc16Chn2,
kAdc16Chn3d = kAdc16Chn3,
kAdc16Chn4a = kAdc16Chn4,
kAdc16Chn5a = kAdc16Chn5,
kAdc16Chn6a = kAdc16Chn6,
kAdc16Chn7a = kAdc16Chn7,
kAdc16Chn4b = kAdc16Chn4,
kAdc16Chn5b = kAdc16Chn5,
kAdc16Chn6b = kAdc16Chn6,
kAdc16Chn7b = kAdc16Chn7 }

```

Defines the type of enumerating ADC16's channel index.

Variables

- `adc16_chn_t adc16_chn_config_t::chnIdx`
Select the sample channel index.
- `bool adc16_chn_config_t::convCompletedIntEnable`
Enable the conversion complete interrupt.
- `bool adc16_converter_config_t::lowPowerEnable`
Enable low power.
- `adc16_clk_divider_t adc16_converter_config_t::clkDividerMode`
Select the divider of input clock source.
- `bool adc16_converter_config_t::longSampleTimeEnable`
Enable the long sample time.
- `adc16_resolution_t adc16_converter_config_t::resolution`
Select the sample resolution mode.
- `adc16_clk_src_mode_t adc16_converter_config_t::clkSrc`
Select the input clock source to converter.
- `bool adc16_converter_config_t::asyncClkEnable`
Enable the asynchronous clock inside the ADC.
- `bool adc16_converter_config_t::highSpeedEnable`
Enable the high speed mode.
- `adc16_long_sample_cycle_t adc16_converter_config_t::longSampleCycleMode`
Select the long sample mode.
- `bool adc16_converter_config_t::hwTriggerEnable`
Enable hardware trigger function.
- `adc16_ref_volt_src_t adc16_converter_config_t::refVoltSrc`
Select the reference voltage source.
- `bool adc16_converter_config_t::continuousConvEnable`
Enable continuous conversion mode.
- `bool adc16_hw_cmp_config_t::hwCmpEnable`
Enable the hardware compare function.
- `bool adc16_hw_cmp_config_t::hwCmpGreaterThanEnable`
Configure the compare function.
- `bool adc16_hw_cmp_config_t::hwCmpRangeEnable`
Configure the comparator function.
- `uint16_t adc16_hw_cmp_config_t::cmpValue1`
Setting value for CV1.
- `uint16_t adc16_hw_cmp_config_t::cmpValue2`
Setting value for CV2.

ADC16 HAL.

- `void ADC16_HAL_Init (ADC_Type *base)`
Resets all registers into a known state for the ADC16 module.
- `void ADC16_HAL_ConfigChn (ADC_Type *base, uint32_t chnGroup, const adc16_chn_config_t *configPtr)`
Configures the conversion channel for the ADC16 module.
- `static bool ADC16_HAL_GetChnConvCompletedFlag (ADC_Type *base, uint32_t chnGroup)`
Checks whether the channel conversion is completed.
- `void ADC16_HAL_ConfigConverter (ADC_Type *base, const adc16_converter_config_t *configPtr)`

ADC16 HAL Driver

- Configures the sampling converter for the ADC16.*

 - void `ADC16_HAL_ConfigHwCompare` (ADC_Type *base, const `adc16_hw_cmp_config_t` *configPtr)
- Configures the hardware comparator function for the ADC16.*

 - static uint16_t `ADC16_HAL_GetChnConvValue` (ADC_Type *base, uint32_t chnGroup)

Gets the raw result data of channel conversion for the ADC16 module.
- static bool `ADC16_HAL_GetConvActiveFlag` (ADC_Type *base)

Checks whether the converter is active for the ADC16 module.

3.2.2 Data Structure Documentation

3.2.2.1 struct `adc16_chn_config_t`

This type of variable is treated as the command to be set in ADC control register, which may execute the ADC's conversion.

Data Fields

- `adc16_chn_t` `chnIdx`
Select the sample channel index.
- bool `convCompletedIntEnable`
Enable the conversion complete interrupt.

3.2.2.2 struct `adc16_converter_config_t`

This type of variable is treated as a group of configurations. Most of the time, these configurations are a one-time setting for converter sampling condition. Usually, they are set before executing the ADC16 job.

Data Fields

- bool `lowPowerEnable`
Enable low power.
- `adc16_clk_divider_t` `clkDividerMode`
Select the divider of input clock source.
- bool `longSampleTimeEnable`
Enable the long sample time.
- `adc16_resolution_t` `resolution`
Select the sample resolution mode.
- `adc16_clk_src_mode_t` `clkSrc`
Select the input clock source to converter.
- bool `asyncClkEnable`
Enable the asynchronous clock inside the ADC.
- bool `highSpeedEnable`
Enable the high speed mode.
- `adc16_long_sample_cycle_t` `longSampleCycleMode`
Select the long sample mode.

- bool `hwTriggerEnable`
Enable hardware trigger function.
- `adc16_ref_volt_src_t refVoltSrc`
Select the reference voltage source.
- bool `continuousConvEnable`
Enable continuous conversion mode.

3.2.2.3 struct `adc16_hw_cmp_config_t`

Data Fields

- bool `hwCmpEnable`
Enable the hardware compare function.
- bool `hwCmpGreaterThanEnable`
Configure the compare function.
- bool `hwCmpRangeEnable`
Configure the comparator function.
- `uint16_t cmpValue1`
Setting value for CV1.
- `uint16_t cmpValue2`
Setting value for CV2.

3.2.3 Enumeration Type Documentation

3.2.3.1 enum `adc16_status_t`

Enumerator

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

3.2.3.2 enum `adc16_clk_divider_t`

Enumerator

- `kAdc16ClkDividerOf1` For divider 1 from the input clock to ADC16.
- `kAdc16ClkDividerOf2` For divider 2 from the input clock to ADC16.
- `kAdc16ClkDividerOf4` For divider 4 from the input clock to ADC16.
- `kAdc16ClkDividerOf8` For divider 8 from the input clock to ADC16.

3.2.3.3 enum `adc16_resolution_t`

Enumerator

- `kAdc16ResolutionBitOf8or9` 8-bit for single end sample, or 9-bit for differential sample.

ADC16 HAL Driver

kAdc16ResolutionBitOfSingleEndAs8 8-bit for single end sample.
kAdc16ResolutionBitOfDiffModeAs9 9-bit for differential sample.
kAdc16ResolutionBitOfI2orI3 12-bit for single end sample, or 13-bit for differential sample.
kAdc16ResolutionBitOfSingleEndAs12 12-bit for single end sample.
kAdc16ResolutionBitOfDiffModeAs13 13-bit for differential sample.
kAdc16ResolutionBitOfI10orI11 10-bit for single end sample, or 11-bit for differential sample.
kAdc16ResolutionBitOfSingleEndAs10 10-bit for single end sample.
kAdc16ResolutionBitOfDiffModeAs11 11-bit for differential sample.

3.2.3.4 enum adc16_clk_src_mode_t

Enumerator

kAdc16ClkSrcOfBusClk For input as bus clock.
kAdc16ClkSrcOfAltClk2 For input as alternate clock 2 (AltClk2).
kAdc16ClkSrcOfAltClk For input as alternate clock (ALTCLK).
kAdc16ClkSrcOfAsynClk For input as asynchronous clock (ADACK).

3.2.3.5 enum adc16_long_sample_cycle_t

Enumerator

kAdc16LongSampleCycleOf24 20 extra ADCK cycles, 24 ADCK cycles total.
kAdc16LongSampleCycleOf16 12 extra ADCK cycles, 16 ADCK cycles total.
kAdc16LongSampleCycleOf10 6 extra ADCK cycles, 10 ADCK cycles total.
kAdc16LongSampleCycleOf4 2 extra ADCK cycles, 6 ADCK cycles total.

3.2.3.6 enum adc16_ref_volt_src_t

Enumerator

kAdc16RefVoltSrcOfVref For external pins pair of VrefH and VrefL.
kAdc16RefVoltSrcOfValt For alternate reference pair of ValtH and ValtL.

3.2.3.7 enum adc16_chn_t

Enumerator

kAdc16Chn0 AD0.
kAdc16Chn1 AD1.
kAdc16Chn2 AD2.
kAdc16Chn3 AD3.

kAdc16Chn4 AD4.
kAdc16Chn5 AD5.
kAdc16Chn6 AD6.
kAdc16Chn7 AD6.
kAdc16Chn8 AD8.
kAdc16Chn9 AD9.
kAdc16Chn10 AD10.
kAdc16Chn11 AD11.
kAdc16Chn12 AD12.
kAdc16Chn13 AD13.
kAdc16Chn14 AD14.
kAdc16Chn15 AD15.
kAdc16Chn16 AD16.
kAdc16Chn17 AD17.
kAdc16Chn18 AD18.
kAdc16Chn19 AD19.
kAdc16Chn20 AD20.
kAdc16Chn21 AD21.
kAdc16Chn22 AD22.
kAdc16Chn23 AD23.
kAdc16Chn24 AD24.
kAdc16Chn25 AD25.
kAdc16Chn26 AD26.
kAdc16Chn27 AD27.
kAdc16Chn28 AD28.
kAdc16Chn29 AD29.
kAdc16Chn30 AD30.
kAdc16Chn31 AD31.
kAdc16Chn0d DAD0.
kAdc16Chn1d DAD1.
kAdc16Chn2d DAD2.
kAdc16Chn3d DAD3.
kAdc16Chn4a AD4a.
kAdc16Chn5a AD5a.
kAdc16Chn6a AD6a.
kAdc16Chn7a AD7a.
kAdc16Chn4b AD4b.
kAdc16Chn5b AD5b.
kAdc16Chn6b AD6b.
kAdc16Chn7b AD7b.

ADC16 HAL Driver

3.2.4 Function Documentation

3.2.4.1 void ADC16_HAL_Init (ADC_Type * *base*)

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

3.2.4.2 void ADC16_HAL_ConfigChn (ADC_Type * *base*, uint32_t *chnGroup*, const adc16_chn_config_t * *configPtr*)

This function configures the channel for the ADC16 module. At any point, only one of the configuration groups takes effect. The other channel group 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, after the available channel is set, the conversion begins to execute.

Parameters

<i>base</i>	Register base address for the module.
<i>chnGroup</i>	Channel configuration group ID.
<i>configPtr</i>	Pointer to configuration structure.

3.2.4.3 static bool ADC16_HAL_GetChnConvCompletedFlag (ADC_Type * *base*, uint32_t *chnGroup*) [inline], [static]

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

Parameters

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

Returns

Assertion of completed conversion mode.

3.2.4.4 void ADC16_HAL_ConfigConverter (ADC_Type * *base*, const adc16_converter_config_t * *configPtr*)

This function configures the sampling converter for the ADC16. Most of the time, the configurations are a one-time setting for the converter sampling condition. Usually, it is called before executing the ADC16 job.

Parameters

<i>base</i>	Register base address for the module.
<i>configPtr</i>	Pointer to configuration structure.

3.2.4.5 void ADC16_HAL_ConfigHwCompare (ADC_Type * *base*, const adc16_hw_cmp_config_t * *configPtr*)

This function configures the hardware comparator function for the ADC16. These are the settings for the ADC16 comparator.

Parameters

<i>base</i>	Register base address for the module.
<i>configPtr</i>	Pointer to configuration structure.

3.2.4.6 static uint16_t ADC16_HAL_GetChnConvValue (ADC_Type * *base*, uint32_t *chnGroup*) [inline], [static]

This function gets the conversion result data for the ADC16 module. The return value is the raw data that is not processed.

Parameters

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

Returns

Conversion value of RAW.

3.2.4.7 static bool ADC16_HAL_GetConvActiveFlag (ADC_Type * *base*) [inline], [static]

This function checks whether the converter is active for the ADC16 module.

ADC16 HAL Driver

Parameters

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

Returns

Assertion of that the converter is active.

3.2.5 Variable Documentation

3.2.5.1 `adc16_chn_t adc16_chn_config_t::chnIdx`

3.2.5.2 `bool adc16_chn_config_t::convCompletedIntEnable`

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:

- **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.

ADC16 Peripheral Driver

- **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.

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.

For One-Time-Trigger use:

```
uint32_t ADC16_DRV_TEST_OneTimeConv(uint32_t instance)
{
    uint32_t errCounter = 0U;
    adc16_converter_config_t userConvConfig;
    adc16_chn_config_t userChnConfig;
    uint16_t sampleValue;
#ifdef FSL_FEATURE_ADC16_HAS_CALIBRATION
    adc16_calibration_param_t userCalConfig;
#endif /* FSL_FEATURE_ADC16_HAS_CALIBRATION */

    /* Execute the auto-calibration. */
    userConvConfig.lowPowerEnable = false;
    userConvConfig.clkDividerMode = kAdc16ClkDividerOf8;
    userConvConfig.longSampleTimeEnable = true;
    userConvConfig.resolution = kAdc16ResolutionBitOf12or13;
    userConvConfig.clkSrc = kAdc16ClkSrcOfAsynClk;
    userConvConfig.asyncClkEnable = true;
    userConvConfig.highSpeedEnable = false;
    userConvConfig.longSampleCycleMode =
        kAdc16LongSampleCycleOf24;
    userConvConfig.hwTriggerEnable = false;
    userConvConfig.refVoltSrc = kAdc16RefVoltSrcOfVref;
    userConvConfig.continuousConvEnable = false;
#ifdef FSL_FEATURE_ADC16_HAS_DMA
    userConvConfig.dmaEnable = false;
#endif /* FSL_FEATURE_ADC16_HAS_DMA */

    if (kStatus_ADC16_Success != ADC16_DRV_Init(instance, &
        userConvConfig))
    {
        errCounter++;
    }

#ifdef FSL_FEATURE_ADC16_HAS_CALIBRATION
    if (kStatus_ADC16_Success != ADC16_DRV_GetAutoCalibrationParam(instance, &
        userCalConfig) )
    {
        errCounter++;
    }
#endif
}
```

```

    ADC16_DRV_SetCalibrationParam(instance, &userCalConfig);
#endif /* FSL_FEATURE_ADC16_HAS_CALIBRATION */

    userChnConfig.chnIdx = ADC16_DRV_TEST_SAMPLE_CHN;
    userChnConfig.convCompletedIntEnable = false;
    userChnConfig.diffConvEnable = false;
    ADC16_DRV_ConfigConvChn(instance, ADC16_DRV_TEST_SAMPLE_CHNGROUP, &userChnConfig
    );

    ADC16_DRV_WaitConvDone(instance, ADC16_DRV_TEST_SAMPLE_CHNGROUP);
    sampleValue = ADC16_DRV_GetConvValueRAW(instance,
        ADC16_DRV_TEST_SAMPLE_CHNGROUP);
    printf("ADC16_DRV_TEST_OneTimeConv Sample Value: %d\r\n", sampleValue);

    ADC16_DRV_Deinit(instance);

    return errCounter;
}

```

For Interrupt use:

```

static uint32_t ADC16_DRV_TEST_OneTimeConvInt(uint32_t instance)
{
    uint32_t errCounter = 0U;
    adc16_converter_config_t userConvConfig;
    adc16_chn_config_t userChnConfig;
    uint16_t sampleValue;

#if FSL_FEATURE_ADC16_HAS_CALIBRATION
    adc16_calibration_param_t userCalConfig;
#endif /* FSL_FEATURE_ADC16_HAS_CALIBRATION */

    /* Execute the auto-calibration. */
    userConvConfig.lowPowerEnable = false;
    userConvConfig.clkDividerMode = kAdc16ClkDividerOf8;
    userConvConfig.longSampleTimeEnable = true;
    userConvConfig.resolution = kAdc16ResolutionBitOf12or13;
    userConvConfig.clkSrc = kAdc16ClkSrcOfAsynClk;
    userConvConfig.asyncClkEnable = true;
    userConvConfig.highSpeedEnable = false;
    userConvConfig.longSampleCycleMode =
        kAdc16LongSampleCycleOf24;
    userConvConfig.hwTriggerEnable = false;
    userConvConfig.refVoltSrc = kAdc16RefVoltSrcOfVref;
    userConvConfig.continuousConvEnable = false;
#if FSL_FEATURE_ADC16_HAS_DMA
    userConvConfig.dmaEnable = false;
#endif /* FSL_FEATURE_ADC16_HAS_DMA */

    if (kStatus_ADC16_Success != ADC16_DRV_Init(instance, &
        userConvConfig))
    {
        errCounter++;
    }

#if FSL_FEATURE_ADC16_HAS_CALIBRATION
    if (kStatus_ADC16_Success != ADC16_DRV_GetAutoCalibrationParam(instance, &
        userCalConfig) )
    {
        errCounter++;
    }
    ADC16_DRV_SetCalibrationParam(instance, &userCalConfig);
#endif /* FSL_FEATURE_ADC16_HAS_CALIBRATION */

    gAdc16IntSampleFlag = false;
}

```

ADC16 Peripheral Driver

```

userChnConfig.chnIdx = ADC16_DRV_TEST_SAMPLE_CHN;
userChnConfig.convCompletedIntEnable = true;
userChnConfig.diffConvEnable = false;
ADC16_DRV_ConfigConvChn(instance, ADC16_DRV_TEST_SAMPLE_CHNGROUP, &userChnConfig
);

while (!gAdc16IntSampleFlag) {}
printf("ADC16_DRV_TEST_OneTimeConvInt Sample Value: %d\r\n", gAdc16IntSampleValue);
gAdc16IntSampleFlag = false;

ADC16_DRV_Deinit(instance);

return errCounter;
}

```

For Blocking use:

```

static uint32_t ADC16_DRV_TEST_ContinuousConvBlocking(uint32_t instance)
{
    uint32_t errCounter = 0U;
    adc16_converter_config_t userConvConfig;
    adc16_chn_config_t userChnConfig;
    uint16_t sampleValue;
    uint32_t i;

#ifdef FSL_FEATURE_ADC16_HAS_CALIBRATION
    adc16_calibration_param_t userCalConfig;
#endif /* FSL_FEATURE_ADC16_HAS_CALIBRATION */

    /* Execute the auto-calibration. */
    userConvConfig.lowPowerEnable = false;
    userConvConfig.clkDividerMode = kAdc16ClkDividerOf8;
    userConvConfig.longSampleTimeEnable = true;
    userConvConfig.resolution = kAdc16ResolutionBitOf12or13;
    userConvConfig.clkSrc = kAdc16ClkSrcOfAsynClk;
    userConvConfig.asyncClkEnable = true;
    userConvConfig.highSpeedEnable = false;
    userConvConfig.longSampleCycleMode =
        kAdc16LongSampleCycleOf24;
    userConvConfig.hwTriggerEnable = false;
    userConvConfig.refVoltSrc = kAdc16RefVoltSrcOfVref;
    userConvConfig.continuousConvEnable = false;
#ifdef FSL_FEATURE_ADC16_HAS_DMA
    userConvConfig.dmaEnable = false;
#endif /* FSL_FEATURE_ADC16_HAS_DMA */

    if (kStatus_ADC16_Success != ADC16_DRV_Init(instance, &
        userConvConfig))
    {
        errCounter++;
    }

#ifdef FSL_FEATURE_ADC16_HAS_CALIBRATION
    if (kStatus_ADC16_Success != ADC16_DRV_GetAutoCalibrationParam(instance, &
        userCalConfig) )
    {
        errCounter++;
    }
    ADC16_DRV_SetCalibrationParam(instance, &userCalConfig);
#endif /* FSL_FEATURE_ADC16_HAS_CALIBRATION */

    userChnConfig.chnIdx = ADC16_DRV_TEST_SAMPLE_CHN;
    userChnConfig.convCompletedIntEnable = false;
    userChnConfig.diffConvEnable = false;

```

```

ADC16_DRV_ConfigConvChn(instance, ADC16_DRV_TEST_SAMPLE_CHNGROUP, &userChnConfig
);

for (i = 0U; i < 4U; i++)
{
    sampleValue = ADC16_DRV_GetConvValueRAW(instance,
ADC16_DRV_TEST_SAMPLE_CHNGROUP);
    printf("ADC16_DRV_TEST_ContinuousConvBlocking %dst Sample Value: %d\r\n", i, sampleValue);
}

ADC16_DRV_Deinit(instance);
return errCounter;
}

```

Enumerations

- enum `adc16_flag_t` {
 - `kAdcConvActiveFlag` = 0U,
 - `kAdcChnConvCompleteFlag` = 3U }*Defines the type of event flags.*

Functions

- `adc16_status_t` `ADC16_DRV_StructInitUserConfigDefault` (`adc16_converter_config_t` *userConfigPtr)
 - Fills the initial user configuration by default for a one-time trigger mode.*
- `adc16_status_t` `ADC16_DRV_Init` (`uint32_t` instance, const `adc16_converter_config_t` *userConfigPtr)
 - Initializes the ADC module converter.*
- `adc16_status_t` `ADC16_DRV_Deinit` (`uint32_t` instance)
 - De-initializes the ADC module converter.*
- `adc16_status_t` `ADC16_DRV_ConfigHwCompare` (`uint32_t` instance, const `adc16_hw_cmp_config_t` *configPtr)
 - Configures the hardware compare feature.*
- `adc16_status_t` `ADC16_DRV_ConfigConvChn` (`uint32_t` instance, `uint32_t` chnGroup, const `adc16_chn_config_t` *configPtr)
 - Configure 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 with no format.*
- `int16_t` `ADC16_DRV_GetConvValueSigned` (`uint32_t` instance, `uint32_t` chnGroup)
 - Gets the latest conversion value with signed.*
- bool `ADC16_DRV_GetConvFlag` (`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.*

ADC16 Peripheral Driver

Variables

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

3.3.5 Enumeration Type Documentation

3.3.5.1 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.

3.3.6 Function Documentation

3.3.6.1 `adc16_status_t ADC16_DRV_StructInitUserConfigDefault (adc16_converter_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:

- `.lowPowerEnable = true;`
- `.clkDividerMode = kAdc16ClkDividerOf8;`
- `.longSampleTimeEnable = true;`
- `.resolutionMode = kAdc16ResolutionBitOfSingleEndAs12;`
- `.clkSrc = kAdc16ClkSrcOfAsynClk`
- `.asynClkEnable = true;`
- `.highSpeedEnable = false;`
- `.longSampleCycleMode = kAdc16LongSampleCycleOf24;`
- `.hwTriggerEnable = false;`
- `.refVoltSrc = kAdcRefVoltSrcOfVref;`
- `.continuousConvEnable = false;`
- `.dmaEnable = false;`

Parameters

<i>userConfigPtr</i>	Pointer to the user configuration structure. See the "adc16_converter_config_t".
----------------------	--

Returns

Execution status.

3.3.6.2 `adc16_status_t ADC16_DRV_Init (uint32_t instance, const adc16_converter_config_t * userConfigPtr)`

This function initializes the converter in the ADC module.

Parameters

<i>instance</i>	ADC16 instance ID.
<i>userConfigPtr</i>	Pointer to the initialization structure. See the "adc16_converter_config_t".

Returns

Execution status.

3.3.6.3 `adc16_status_t 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.
-----------------	--------------------

Returns

Execution status.

3.3.6.4 `adc16_status_t ADC16_DRV_ConfigHwCompare (uint32_t instance, const adc16_hw_cmp_config_t * configPtr)`

This function configures the hardware compare feature with indicated configuration.

ADC16 Peripheral Driver

Parameters

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

Returns

Execution status.

3.3.6.5 `adc16_status_t` **ADC16_DRV_ConfigConvChn** (`uint32_t instance`, `uint32_t chnGroup`, `const adc16_chn_config_t * configPtr`)

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>configPtr</i>	Pointer to configuration structure. See the "adc16_chn_config_t".

Returns

Execution status.

3.3.6.6 `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.

Parameters

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

<i>chnGroup</i>	Selection of configuration group.
-----------------	-----------------------------------

3.3.6.7 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.

3.3.6.8 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.

3.3.6.9 int16_t ADC16_DRV_GetConvValueSigned (uint32_t *instance*, uint32_t *chnGroup*)

This function gets the conversion value from the ADC16 module with signed.

Parameters

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

Returns

Signed conversion value.

ADC16 Peripheral Driver

3.3.6.10 `bool ADC16_DRV_GetConvFlag (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.

3.3.6.11 **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.

3.3.7 Variable Documentation

3.3.7.1 **ADC_Type* const g_adcBase[]**

3.3.7.2 **const IRQn_Type g_adcIrqId[ADC_INSTANCE_COUNT]**

Chapter 4

Analog Front End (AFE)

4.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Analog Front End (AFE) block of Kinetis devices.

Modules

- [AFE HAL driver](#)
- [AFE Peripheral driver](#)

AFE HAL driver

4.2 AFE HAL driver

4.2.1 Overview

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

Data Structures

- struct [afe_chn_set_t](#)
Defines the structure to configure the AFE Channel. [More...](#)
- struct [afe_converter_config_t](#)
Defines the structure to configure the AFE converter. [More...](#)

Enumerations

- enum [afe_status_t](#) {
 [kStatus_AFE_Success](#) = 0U,
 [kStatus_AFE_InvalidArgument](#) = 1U,
 [kStatus_AFE_Failed](#) = 2U }
AFE status return codes.
- enum [afe_chn_osr_mode_t](#) {
 [kAfeDecimOsrOf64](#) = 0U,
 [kAfeDecimOsrOf128](#) = 1U,
 [kAfeDecimOsrOf256](#) = 2U,
 [kAfeDecimOsrOf512](#) = 3U,
 [kAfeDecimOsrOf1024](#) = 4U,
 [kAfeDecimOsrOf2048](#) = 5U }
AFE OSR modes.
- enum [afe_pga_gain_mode_t](#) {
 [kAfePgaGainBy1](#) = 1U,
 [kAfePgaGainBy2](#) = 2U,
 [kAfePgaGainBy4](#) = 3U,
 [kAfePgaGainBy8](#) = 4U,
 [kAfePgaGainBy16](#) = 5U,
 [kAfePgaGainBy32](#) = 6U }
AFE PGA modes.
- enum [afe_result_format_mode_t](#) {
 [kAfeResultFormatLeft](#) = 0U,
 [kAfeResultFormatRight](#) = 1U }
Defines the AFE result format modes.
- enum [afe_clk_divider_mode_t](#) {

```

kAfeClkDividerInputOf1 = 0U,
kAfeClkDividerInputOf2 = 1U,
kAfeClkDividerInputOf4 = 2U,
kAfeClkDividerInputOf8 = 3U,
kAfeClkDividerInputOf16 = 4U,
kAfeClkDividerInputOf32 = 5U,
kAfeClkDividerInputOf64 = 6U,
kAfeClkDividerInputOf128 = 7U,
kAfeClkDividerInputOf256 = 8U }

```

Defines the AFE clock divider modes.

- enum `afe_clk_src_mode_t` {

```

kAfeClkSrcClk0 = 0U,
kAfeClkSrcClk1 = 1U,
kAfeClkSrcClk2 = 2U,
kAfeClkSrcClk3 = 3U }

```

Defines the AFE clock source modes.

AFE Channel Configurations

- void `AFE_HAL_ChnInit` (AFE_Type *base, uint32_t chn)
 - Resets the channel registers into a reset state.*
- static void `AFE_HAL_SetIntEnableCmd` (AFE_Type *base, uint32_t chn, bool enable)
 - Sets the interrupt request on conversion complete event.*
- static void `AFE_HAL_SetDmaEnableCmd` (AFE_Type *base, uint32_t chn, bool enable)
 - Sets the DMA request on conversion complete event.*
- static void `AFE_HAL_SetDelayVal` (AFE_Type *base, uint32_t chn, int32_t delay)
 - Sets the delay value.*
- static int32_t `AFE_HAL_GetDelayVal` (AFE_Type *base, uint32_t chn)
 - Gets the delay value.*
- static uint32_t `AFE_HAL_GetResult` (AFE_Type *base, uint32_t chn)
 - Gets the measurement result.*
- static bool `AFE_HAL_GetConvCompleteFlag` (AFE_Type *base, uint32_t chn)
 - Checks whether a conversion is completed.*
- static bool `AFE_HAL_GetOverflowFlag` (AFE_Type *base, uint32_t chn)
 - Checks whether the selected channel has overflowed.*
- static bool `AFE_HAL_GetReadyFlag` (AFE_Type *base, uint32_t chn)
 - Checks whether the selected channel is ready for conversion.*

AFE Common Configurations

- void `AFE_HAL_Init` (AFE_Type *base)
 - Resets all registers into a reset state for the AFE module.*
- static void `AFE_HAL_SetMasterEnableCmd` (AFE_Type *base, bool enable)
 - Enables all ADCs and filters.*
- static void `AFE_HAL_ChnSwTriggerCmd` (AFE_Type *base, uint32_t chnMask)
 - Triggers conversion on the AFE channels.*
- static void `AFE_HAL_SetLowPowerCmd` (AFE_Type *base, bool enable)

AFE HAL driver

- *Sets a low power mode.*
- static void [AFE_HAL_SetSwResetCmd](#) (AFE_Type *base, bool enable)
- *Sets the software reset.*
- static void [AFE_HAL_SetDelayOkCmd](#) (AFE_Type *base)
- *Asserts the delay OK.*
- static void [AFE_HAL_SetResultFormatMode](#) (AFE_Type *base, [afe_result_format_mode_t](#) mode)
- *Selects the result format mode.*
- static [afe_result_format_mode_t](#) [AFE_HAL_GetResultFormatMode](#) (AFE_Type *base)
- *Gets the conversion resolution mode.*
- static void [AFE_HAL_SetStartUpDelayVal](#) (AFE_Type *base, int32_t startDelay)
- *Sets the start up count value.*
- static void [AFE_HAL_SetClkDividerMode](#) (AFE_Type *base, [afe_clk_divider_mode_t](#) mode)
- *Selects the clock divider mode.*
- static void [AFE_HAL_SetClkSourceMode](#) (AFE_Type *base, [afe_clk_src_mode_t](#) mode)
- *Selects the input clock source for the AFE module.*
- static void [AFE_HAL_SetDmaIntReq](#) (AFE_Type *base, uint32_t data)
- *Sets the DMA and interrupt requests for all channels.*
- void [AFE_HAL_ConfigChn](#) (AFE_Type *base, uint32_t chn, [afe_chn_set_t](#) *chnStructPtr)
- *Configures the AFE channel.*
- void [AFE_HAL_ConfigConverter](#) (AFE_Type *base, [afe_converter_config_t](#) *converterStructPtr)
- *Configures the AFE converter.*

4.2.2 Data Structure Documentation

4.2.2.1 struct [afe_chn_set_t](#)

Data Fields

- bool [hwTriggerEnable](#)
Enable triggering by hardware.
- bool [continuousConvEnable](#)
Enable continuous conversion mode.
- bool [bypassEnable](#)
Enable bypass mode that modulator and PGA of this channel are disabled.
- [afe_pga_gain_mode_t](#) [pgaGainSel](#)
Select the analog gain applied to the input signal.
- bool [pgaEnable](#)
Enable the analog gain.
- [afe_chn_osr_mode_t](#) [decimOSR](#)
Select the over sampling ration.
- bool [modulatorEnable](#)
Enable sigma delta modulator.
- bool [decimFilterEnable](#)
Enable decimation filter.
- bool [decimNedgeEnable](#)
Enable falling edge for decimator input.
- bool [externClockEnable](#)
Enable external clock for decimator.

4.2.2.1.0.1 Field Documentation

4.2.2.1.0.1.1 `bool afe_chn_set_t::hwTriggerEnable`

4.2.2.1.0.1.2 `bool afe_chn_set_t::continuousConvEnable`

4.2.2.1.0.1.3 `bool afe_chn_set_t::bypassEnable`

4.2.2.1.0.1.4 `afe_pga_gain_mode_t afe_chn_set_t::pgaGainSel`

4.2.2.1.0.1.5 `bool afe_chn_set_t::pgaEnable`

4.2.2.1.0.1.6 `afe_chn_osr_mode_t afe_chn_set_t::decimOSR`

4.2.2.1.0.1.7 `bool afe_chn_set_t::modulatorEnable`

4.2.2.2 `struct afe_converter_config_t`

Data Fields

- `uint8_t startupCnt`
Set startup delay of modulators.
- `afe_result_format_mode_t resultFormat`
Select the result format.
- `bool delayOk`
Confirm delay registers.
- `bool swReset`
Do a software reset.
- `bool lowPowerEnable`
Enable low power mode.
- `bool swTriggChn0`
Trigger conversion on channel 0.
- `bool swTriggChn1`
Trigger conversion on channel 1.
- `bool swTriggChn2`
Trigger conversion on channel 2.
- `bool masterEnable`
Enable all ADCs and filters simultaneously whose modulatorEnable and decimFilterEnable is asserted.

AFE HAL driver

4.2.2.2.0.2 Field Documentation

4.2.2.2.0.2.1 `uint8_t afe_converter_config_t::startupCnt`

4.2.2.2.0.2.2 `afe_result_format_mode_t afe_converter_config_t::resultFormat`

4.2.2.2.0.2.3 `bool afe_converter_config_t::delayOk`

4.2.2.2.0.2.4 `bool afe_converter_config_t::swReset`

4.2.2.2.0.2.5 `bool afe_converter_config_t::lowPowerEnable`

4.2.2.2.0.2.6 `bool afe_converter_config_t::swTriggChn0`

4.2.2.2.0.2.7 `bool afe_converter_config_t::swTriggChn1`

4.2.2.2.0.2.8 `bool afe_converter_config_t::swTriggChn2`

4.2.2.2.0.2.9 `bool afe_converter_config_t::masterEnable`

4.2.3 Enumeration Type Documentation

4.2.3.1 `enum afe_status_t`

Enumerator

kStatus_AFE_Success Success.

kStatus_AFE_InvalidArgument Invalid argument existed.

kStatus_AFE_Failed Execution failed.

4.2.3.2 `enum afe_chn_osr_mode_t`

Enumerator

kAfeDecimOsrOf64 Decimator over sampling ratio is 64.

kAfeDecimOsrOf128 Decimator over sampling ratio is 128.

kAfeDecimOsrOf256 Decimator over sampling ratio is 256.

kAfeDecimOsrOf512 Decimator over sampling ratio is 512.

kAfeDecimOsrOf1024 Decimator over sampling ratio is 1024.

kAfeDecimOsrOf2048 Decimator over sampling ratio is 2048.

4.2.3.3 `enum afe_pga_gain_mode_t`

Enumerator

kAfePgaGainBy1 Input gained by 1.

kAfePgaGainBy2 Input gained by 2.

kAfePgaGainBy4 Input gained by 4.
kAfePgaGainBy8 Input gained by 8.
kAfePgaGainBy16 Input gained by 16.
kAfePgaGainBy32 Input gained by 32.

4.2.3.4 enum afe_result_format_mode_t

Enumerator

kAfeResultFormatLeft Left justified result format.
kAfeResultFormatRight Right justified result format.

4.2.3.5 enum afe_clk_divider_mode_t

Enumerator

kAfeClkDividerInputOf1 Clock divided by 1.
kAfeClkDividerInputOf2 Clock divided by 2.
kAfeClkDividerInputOf4 Clock divided by 4.
kAfeClkDividerInputOf8 Clock divided by 8.
kAfeClkDividerInputOf16 Clock divided by 16.
kAfeClkDividerInputOf32 Clock divided by 32.
kAfeClkDividerInputOf64 Clock divided by 64.
kAfeClkDividerInputOf128 Clock divided by 128.
kAfeClkDividerInputOf256 Clock divided by 256.

4.2.3.6 enum afe_clk_src_mode_t

Enumerator

kAfeClkSrcClk0 Modulator clock source 0.
kAfeClkSrcClk1 Modulator clock source 1.
kAfeClkSrcClk2 Modulator clock source 2.
kAfeClkSrcClk3 Modulator clock source 3.

4.2.4 Function Documentation

4.2.4.1 void AFE_HAL_ChnInit (AFE_Type * base, uint32_t chn)

This function resets the channel registers into a reset state. This state is defined in the chip Reference Manual.

AFE HAL driver

Parameters

<i>base</i>	Register base address for the module.
<i>chn</i>	AFE Channel.

4.2.4.2 `static void AFE_HAL_SetIntEnableCmd (AFE_Type * base, uint32_t chn, bool enable) [inline], [static]`

This function sets the interrupt request on conversion complete event. Once enabled, the conversion complete interrupt request is asserted for the desired channel when INTENx and COCx bits are set.

Parameters

<i>base</i>	Register base address for the AFE module.
<i>chn</i>	AFE Channel.
<i>enable</i>	Bool value for enabling or disabling interrupt function.

4.2.4.3 `static void AFE_HAL_SetDmaEnableCmd (AFE_Type * base, uint32_t chn, bool enable) [inline], [static]`

This function sets the DMA request on conversion complete event. Once enabled, the DMA request is asserted when the INTENx and the COCx bits are set.

Parameters

<i>base</i>	Register base address for the AFE module.
<i>chn</i>	AFE Channel.
<i>enable</i>	Bool value for enabling or disabling DMA function.

4.2.4.4 `static void AFE_HAL_SetDelayVal (AFE_Type * base, uint32_t chn, int32_t delay) [inline], [static]`

This function sets the delay value. This delay is inserted into the trigger response of the decimation filters in order to provide phase compensation between AFE channels.

Parameters

<i>base</i>	Register base address for the AFE module.
<i>chn</i>	AFE Channel.
<i>delay</i>	Delay value.

4.2.4.5 **static int32_t AFE_HAL_GetDelayVal (AFE_Type * *base*, uint32_t *chn*)** **[inline], [static]**

This function gets the delay value.

Parameters

<i>base</i>	Register base address for the AFE module.
<i>chn</i>	AFE Channel.

Returns

Delay value.

4.2.4.6 **static uint32_t AFE_HAL_GetResult (AFE_Type * *base*, uint32_t *chn*)** **[inline], [static]**

This function gets the measurement result. The result can be represented in left justified 2's complement 32-bit or right justified 2's complement 32-bit format. See the "AFE_HAL_SetResultFormatMode()" for format setting.

Parameters

<i>base</i>	Register base address for the AFE module.
<i>chn</i>	AFE Channel.

Returns

result.

4.2.4.7 **static bool AFE_HAL_GetConvCompleteFlag (AFE_Type * *base*, uint32_t *chn*)** **[inline], [static]**

This function checks whether the conversion is completed. When the result is read (see the "AFE_HAL_GetResult()"), the corresponding bit is cleared.

AFE HAL driver

Parameters

<i>base</i>	Register base address for the AFE module.
<i>chn</i>	AFE Channel.

Returns

Conversion is completed (true) or not (false).

4.2.4.8 static bool AFE_HAL_GetOverflowFlag (AFE_Type * *base*, uint32_t *chn*) [inline], [static]

This function checks whether the selected channel has overflow. This function returns true when new data has already been arrived but previous data has not been read. The corresponding bit is cleared when the result is read (see the "AFE_HAL_GetResult()").

Parameters

<i>base</i>	Register base address for the AFE module.
<i>chn</i>	AFE Channel.

Returns

Channel is overflowed (true) or not (false).

4.2.4.9 static bool AFE_HAL_GetReadyFlag (AFE_Type * *base*, uint32_t *chn*) [inline], [static]

This function checks whether the selected channel is ready for conversion.

Parameters

<i>base</i>	Register base address for the AFE module.
<i>chn</i>	AFE Channel.

Returns

Channel is ready to initiate conversions (true) or is disabled or has not completed its start up period (false).

4.2.4.10 void AFE_HAL_Init (AFE_Type * *base*)

This function resets all registers into a reset state for the AFE module. This state is defined in the chip Reference Manual.

Parameters

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

4.2.4.11 static void AFE_HAL_SetMasterEnableCmd (AFE_Type * *base*, bool *enable*) [inline], [static]

This function enables all ADCs and filters. The only ADCs that have asserted are SD_MOD_EN (only when AFE channel is not bypassed) and DEC_EN () are enabled by setting this bit.

Parameters

<i>base</i>	Register base address for the AFE module.
<i>enable</i>	Bool value for enabling or disabling ADCs and filters simultaneously.

4.2.4.12 static void AFE_HAL_ChnSwTriggerCmd (AFE_Type * *base*, uint32_t *chnMask*) [inline], [static]

This function sets the software trigger conversion on the desired AFE channels. To trigger more channels simultaneously, use the ored enumeration members.

Parameters

<i>base</i>	Register base address for the AFE module.
<i>chns</i>	Channels which will be triggered.

4.2.4.13 static void AFE_HAL_SetLowPowerCmd (AFE_Type * *base*, bool *enable*) [inline], [static]

This function sets an AFE low power mode. The maximal modulator clock in low power mode is below 1 MHz.

Parameters

<i>base</i>	Register base address for the AFE module.
<i>enable</i>	Bool value for enabling or disabling low power mode.

AFE HAL driver

4.2.4.14 `static void AFE_HAL_SetSwResetCmd (AFE_Type * base, bool enable)`
`[inline], [static]`

This function sets the software reset which is used to reset all ADCs, decimation filters, and clock configuration bits.

Parameters

<i>base</i>	Register base address for the AFE module.
<i>enable</i>	If this value is true, all ADCs, PGAs and Decimation filters are enabled otherwise are disabled.

4.2.4.15 static void AFE_HAL_SetDelayOkCmd (AFE_Type * *base*) [inline], [static]

This function asserts the delay OK. When all delay registers are loaded, the delay OK bit should be asserted.

Parameters

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

4.2.4.16 static void AFE_HAL_SetResultFormatMode (AFE_Type * *base*, *afe_result_format_mode_t mode*) [inline], [static]

This function selects the result format mode.

Parameters

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

4.2.4.17 static *afe_result_format_mode_t* AFE_HAL_GetResultFormatMode (AFE_Type * *base*) [inline], [static]

This function gets the conversion resolution mode.

Parameters

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

Returns

Current conversion resolution mode.

AFE HAL driver

4.2.4.18 `static void AFE_HAL_SetStartUpDelayVal (AFE_Type * base, int32_t startDelay) [inline], [static]`

This function sets the start up count value. A minimum value is two. This value should be greater than 20 μ s. To compute the 20 μ s value use, such as $startDelay = (CLOCK_FREQ / DIV_factor) * 20E-6$

Parameters

<i>base</i>	Register base address for the AFE module.
<i>startDelay</i>	Value for start up count.

**4.2.4.19 static void AFE_HAL_SetClkDividerMode (AFE_Type * *base*,
afe_clk_divider_mode_t *mode*) [inline], [static]**

This function selects the clock divider mode.

Parameters

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

**4.2.4.20 static void AFE_HAL_SetClkSourceMode (AFE_Type * *base*,
afe_clk_src_mode_t *mode*) [inline], [static]**

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

Parameters

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

**4.2.4.21 static void AFE_HAL_SetDmaIntReq (AFE_Type * *base*, uint32_t *data*)
[inline], [static]**

This function sets a DMA and Interrupt requests for all channels.

Parameters

<i>base</i>	Register base address for the AFE module.
<i>data</i>	which will be written to register.

**4.2.4.22 void AFE_HAL_ConfigChn (AFE_Type * *base*, uint32_t *chn*, afe_chn_set_t *
chnStructPtr)**

This function configures the AFE channel.

AFE HAL driver

Parameters

<i>base</i>	Register base address for the AFE module.
<i>chn</i>	AFE Channel.
<i>chnStructPtr</i>	Channel configuration structure.

4.2.4.23 void AFE_HAL_ConfigConverter (AFE_Type * *base*, afe_converter_config_t * *converterStructPtr*)

This function configures an AFE converter.

Parameters

<i>base</i>	Register base address for the AFE module.
<i>convStructPtr</i>	Converter configuration structure.

4.3 AFE Peripheral driver

4.3.1 Overview

This section describes the programming interface of the AFE Peripheral driver. The AFE peripheral driver provides functions to initialize AFE module and channels, conversion triggering, and result reading.

4.3.2 Channel configuration structures

The driver uses instances of the channel configuration structures to configuration and initialization AFE channel. This structure holds the settings of the AFE measurement channel. The settings include AFE hardware/software triggering, AFE continuous/Single conversion mode, AFE phase delay compensation, AFE channel mode, AFE channel analog gain, AFE channel oversampling ration, and AFE channel interrupt or DMA function. The AFE channel mode selects whether the bypass mode is enabled or disabled and the external clock selection.

4.3.3 User configuration structures

The AFE driver uses instances of the user configuration structure `afe_user_config_t` for the AFE driver configuration. This structure holds the configuration which is common for all AFE channels. The settings include AFE low power mode, AFE result format, AFE clock divider mode, AFE clock source mode, and AFE start up delay of modulators.

4.3.4 Initialization

1. To initialize the AFE driver, call the `AFE_DRV_Init()` function and pass the instance number of the AFE peripheral and a pointer to the user configuration structure. For a typical use case, call the `AFE_DRV_StructInitUserConfigDefault()` function which populates the structure.
2. Then, call the `AFE_DRV_ChnInit` function and pass the instance number, channel number and a pointer to the channel configuration structure. For a typical use case, call the `AFE_DRV_StructInitChnConfigDefault()` function which populates the structure.
3. The start up value can be set by the `AFE_DRV_SetStartUpVal20us()` function call or by filling the `startupCnt` parameter of the `AfeChnConfig` structure.
4. If the phase delay values are configured, call the `AFE_DRV_AssertDelayOk()` function and pass the instance number.
5. Finally, call the `AFE_DRV_Enable()` function and pass the instance number and a true boolean parameter. Ensure that the clock and VREF are configured properly before configuring the AFE. For more information, see a VREF and documentation for the KSDK Clock Manager.

```
typedef struct AfeUserConfig
{
    bool lowPowerEnable;
    afe_result_format_mode_t resultFormat;
```

AFE Peripheral driver

```

afe_clk_divider_mode_t clkDividerMode;
afe_clk_src_mode_t clkSrcMode;
uint8_t startupCnt;
} afe_user_config_t;

typedef struct AfeChnConfig
{
    bool hwTriggerEnable;
    bool continuousConvEnable;
    int32_t delay;
    afe_chn_mode_t chnMode;
    afe_pga_state_t pgaGainSel;
    afe_chn_osr_mode_t decimOSR;
    afe_chn_event_t chnEvent;
} afe_chn_config_t;

```

This is an example of a typical AFE configuration. The `afe_user_config_t` instantiation is configured in the normal mode (low power mode disabled), result format is justified to the right, clock divider is selected 2 and a first clock source is selected (see the microcontroller documentation for information about these selections). The `afe_chn_config_t` instantiation is configured in the normal mode (no bypass), a continuous conversion, a zero delay, no interrupt or DMA request, the software trigger enabled, and the PGA disabled and the OSR is 2048.

This is an example code to set up a user and a channel AFE configuration instantiation:

```

afe_user_config_t afeConfig;
afeConfig.clkDividerMode = kAfeClkDividerInputOf2;
afeConfig.clkSrcMode = kAfeClkSrcClk1;
afeConfig.lowPowerEnable = false;
afeConfig.resultFormat = kAfeResultFormatRight;
afeTestStruct.startupCnt = 80; // startupCnt = (Clk_freq/Clk_div)*20e-6

afe_chn_config_t afeChnConfig;
afeChnConfig.chnMode = kAfeNormal;
afeChnConfig.continuousConvEnable = true;
afeChnConfig.delay = 0;
afeChnConfig.chnEvent = kAfeNoneReq;
afeChnConfig.hwTriggerEnable = false;
afeChnConfig.pgaGainSel = kAfePgaDisable;
afeChnConfig.decimOSR = kAfeDecimOsrOf2048;

```

This example shows how to call the `AFE_DRV_Init()` given the user configuration structure and the AFE instance 0.

```

uint32_t afeInstance = 0;
UART_DRV_Init(afeInstance, &afeConfig);

```

This example shows how to call the `AFE_DRV_ChnInit()` given the channel configuration structure and the AFE channel 0 and 1.

```

uint32_t afeChannel0 = 0;
uint32_t afeChannel1 = 1;
AFE_DRV_ChnInit(afeInstance, afeChannel0, &afeChnConfig); // Configure AFE channel 0
AFE_DRV_ChnInit(afeInstance, afeChannel1, &afeChnConfig); // Configure AFE channel 1

```

The last part of the initialization contains the start up count setting, delay values confirmation, and the AFE module enabling.

```

AFE_DRV_AssertDelayOk(afeInstance); // Assert delay for all channels simultaneously
AFE_DRV_Enable(afeInstance, true); // Enable all configured channels

```

4.3.5 Measuring

The driver contains functions for software triggering, a channel delay after trigger setting, a result (raw or converted to right justified), reading and waiting functions.

If the software triggering is enabled (hwTriggerEnable parameter in afe_chn_config is a false value), call the [AFE_DRV_SoftTriggerConv\(\)](#) function to start conversion. In this example, the channel 0 and channel 1 are triggered simultaneously. After the conversion is completed, the results are read.

```
int32_t res0, res1;

AFE_DRV_SoftTriggerConv(afeInstance, CHN_TRIG_MASK(0) | CHN_TRIG_MASK(1));

while(1)
{
    AFE_DRV_WaitConvDone(afeInstance, 0);
    AFE_DRV_WaitConvDone(afeInstance, 1);

    res0 = AFE_DRV_GetChnConvValue(afeInstance, 0);
    res1 = AFE_DRV_GetChnConvValue(afeInstance, 1);
}
```

If the hardware triggering is enabled (hwTriggerEnable parameter in afe_chn_config is a true value), the [AFE_DRV_SoftTriggerConv\(\)](#) function isn't called. The rest of the code is the same.

If the continuous conversion isn't allowed (continuousConvEnable parameter in afe_chn_config is a false value), call the [AFE_DRV_SoftTriggerConv\(\)](#) function (in the software trigger case) in the main loop to trigger the next conversion.

Data Structures

- struct [afe_chn_config_t](#)
Defines the structure to initialize the AFE channel. [More...](#)
- struct [afe_user_config_t](#)
Defines the structure to initialize the AFE module. [More...](#)
- struct [afe_delay_config_t](#)
Defines the structure to configure phase delay of each AFE channel. [More...](#)

Enumerations

- enum [afe_pga_state_t](#) {
[kAfePgaDisable](#) = 0U,
[kAfePgaGain1](#) = 1U,
[kAfePgaGain2](#) = 2U,
[kAfePgaGain4](#) = 3U,
[kAfePgaGain8](#) = 4U,
[kAfePgaGain16](#) = 5U,
[kAfePgaGain32](#) = 6U }
Defines the PGA's values.

AFE Peripheral driver

- enum `afe_chn_mode_t` {
`kAfeNormal` = 0,
`kAfeBypassExternClkPosEdge` = 1,
`kAfeBypassExternClkNegEdge` = 2,
`kAfeBypassInternClkPosEdge` = 3,
`kAfeBypassInternClkNegEdge` = 4 }
Defines the channel's modes.
- enum `afe_chn_event_t` {
`kAfeNoneReq` = 0,
`kAfeIntReq` = 1,
`kAfeDmaReq` = 2 }
Defines the channel's event requests.
- enum `afe_flag_t` {
`kAfeOverflowFlag` = 0U,
`kAfeReadyFlag` = 1U,
`kAfeConvCompleteFlag` = 2U }
Defines the type of event flags.

Functions

- `afe_status_t AFE_DRV_StructInitUserConfigDefault` (`afe_user_config_t *userConfigPtr`)
Fills the user configure structure.
- `afe_status_t AFE_DRV_StructInitChnConfigDefault` (`afe_chn_config_t *chnConfigPtr`)
Fills the channel configuration structure.
- `afe_status_t AFE_DRV_Init` (`uint32_t instance`, `afe_user_config_t *userConfigPtr`)
Initializes the AFE module.
- `afe_status_t AFE_DRV_ChnInit` (`uint32_t instance`, `uint32_t chn`, `afe_chn_config_t *chnConfigPtr`)
Initializes the selected AFE channel.
- void `AFE_DRV_Enable` (`uint32_t instance`, bool enable)
Enables/disables all configured AFE channels.
- void `AFE_DRV_WaitConvDone` (`uint32_t instance`, `uint32_t chn`)
Waits until the last conversion is complete.
- void `AFE_DRV_WaitChnReady` (`uint32_t instance`, `uint32_t chn`)
Waits until the channel is ready for conversion.
- void `AFE_DRV_SoftTriggerConv` (`uint32_t instance`, `uint32_t chnMask`)
Triggers the AFE conversion by software.
- bool `AFE_DRV_GetChnFlag` (`uint32_t instance`, `uint32_t chn`, `afe_flag_t flag`)
Gets the flag for channel's events.
- `uint32_t AFE_DRV_GetChnConvValRaw` (`uint32_t instance`, `uint32_t chn`)
Reads the conversion value in a raw form.
- `int32_t AFE_DRV_GetChnConvVal` (`uint32_t instance`, `uint32_t chn`)
Reads the conversion value in 2's complement form.
- void `AFE_DRV_Deinit` (`uint32_t instance`)
De-initializes the AFE module.
- void `AFE_DRV_ChnDeinit` (`uint32_t instance`, `uint32_t chn`)
De-initializes the selected AFE channel.
- void `AFE_DRV_AssertDelayOk` (`uint32_t instance`)
Asserts the phase delay setting.

- void [AFE_DRV_SetPhaseDelays](#) (uint32_t instance, [afe_delay_config_t](#) *delayConfigPtr)
Sets phase delays.

Variables

- AFE_Type *const [g_afeBase](#) []
Table of base addresses for AFE instances.
- const IRQn_Type [g_afeIrqId](#) []
Table to save AFE IRQ enumeration numbers defined in CMSIS header file.

4.3.6 Data Structure Documentation

4.3.6.1 struct [afe_chn_config_t](#)

This structure keeps the configuration for the AFE channel.

Data Fields

- bool [hwTriggerEnable](#)
Enable triggering by hardware.
- bool [continuousConvEnable](#)
Enable continuous conversion mode.
- int32_t [delay](#)
Set the phase compensation.
- [afe_chn_mode_t](#) [chnMode](#)
Select if channel is in bypassed mode.
- [afe_pga_state_t](#) [pgaGainSel](#)
Select the analog gain applied to the input signal.
- [afe_chn_osr_mode_t](#) [decimOSR](#)
Select the over sampling ration.
- [afe_chn_event_t](#) [chnEvent](#)
Select DMA or interrupt function.

AFE Peripheral driver

4.3.6.1.0.3 Field Documentation

4.3.6.1.0.3.1 `bool afe_chn_config_t::hwTriggerEnable`

4.3.6.1.0.3.2 `bool afe_chn_config_t::continuousConvEnable`

4.3.6.1.0.3.3 `int32_t afe_chn_config_t::delay`

4.3.6.1.0.3.4 `afe_chn_mode_t afe_chn_config_t::chnMode`

4.3.6.1.0.3.5 `afe_pga_state_t afe_chn_config_t::pgaGainSel`

4.3.6.1.0.3.6 `afe_chn_osr_mode_t afe_chn_config_t::decimOSR`

4.3.6.1.0.3.7 `afe_chn_event_t afe_chn_config_t::chnEvent`

4.3.6.2 struct `afe_user_config_t`

This structure keeps the configuration for the AFE module.

Data Fields

- `bool lowPowerEnable`
Enable low power mode.
- `afe_result_format_mode_t resultFormat`
Select the result format.
- `afe_clk_divider_mode_t clkDividerMode`
Select the clock divider ration for the modulator clock.
- `afe_clk_src_mode_t clkSrcMode`
Select clock source for modulator clock.
- `uint8_t startupCnt`
Select the start up delay of modulators.

4.3.6.2.0.4 Field Documentation

4.3.6.2.0.4.1 `bool afe_user_config_t::lowPowerEnable`

4.3.6.2.0.4.2 `afe_result_format_mode_t afe_user_config_t::resultFormat`

4.3.6.2.0.4.3 `afe_clk_divider_mode_t afe_user_config_t::clkDividerMode`

4.3.6.2.0.4.4 `afe_clk_src_mode_t afe_user_config_t::clkSrcMode`

4.3.6.2.0.4.5 `uint8_t afe_user_config_t::startupCnt`

4.3.6.3 struct `afe_delay_config_t`

This structure keeps the phase delay of each AFE channel.

Data Fields

- uint32_t `delayChn0`
Phase compensation of channel0.
- uint32_t `delayChn1`
Phase compensation of channel1.
- uint32_t `delayChn2`
Phase compensation of channel2.

4.3.6.3.0.5 Field Documentation

4.3.6.3.0.5.1 uint32_t afe_delay_config_t::delayChn0

4.3.6.3.0.5.2 uint32_t afe_delay_config_t::delayChn1

4.3.6.3.0.5.3 uint32_t afe_delay_config_t::delayChn2

4.3.7 Enumeration Type Documentation

4.3.7.1 enum afe_pga_state_t

Enumerator

- kAfePgaDisable* Pga disabled.
- kAfePgaGain1* Input gained by 1.
- kAfePgaGain2* Input gained by 2.
- kAfePgaGain4* Input gained by 4.
- kAfePgaGain8* Input gained by 8.
- kAfePgaGain16* Input gained by 16.
- kAfePgaGain32* Input gained by 32.

4.3.7.2 enum afe_chn_mode_t

Enumerator

- kAfeNormal* Normal channel mode.
- kAfeBypassExternClkPosEdge* Bypassed channel mode - external clock selected, positive edge for registering data by the decimation filter.
- kAfeBypassExternClkNegEdge* Bypassed channel mode - external clock selected, negative edge for registering data by the decimation filter.
- kAfeBypassInternClkPosEdge* Bypassed channel mode - internal clock selected, positive edge for registering data by the decimation filter.
- kAfeBypassInternClkNegEdge* Bypassed channel mode - external clock selected, negative edge for registering data by the decimation filter.

AFE Peripheral driver

4.3.7.3 enum afe_chn_event_t

Enumerator

kAfeNoneReq None request is enabled if conversion is completed.

kAfeIntReq Interrupt request is enabled if conversion is completed.

kAfeDmaReq DMA request is enabled if conversion is completed.

4.3.7.4 enum afe_flag_t

Enumerator

kAfeOverflowFlag Indicates if a previous conversion result has not been read and new data has already been arrived.

kAfeReadyFlag Indicates if a channel is ready to conversion.

kAfeConvCompleteFlag Indicates if a conversion is completed.

4.3.8 Function Documentation

4.3.8.1 afe_status_t AFE_DRV_StructInitUserConfigDefault (afe_user_config_t * userConfigPtr)

This function fills the [afe_user_config_t](#) structure with default settings. These setting are:

```
.lowPowerEnable = false
.resultFormat = kAfeResultFormatLeft
.clkDividerMode = kAfeClkDividerInputOf2
.clkSrcMode = kAfeClkSrcClk0
.startupCnt = 125
```

Parameters

<i>userConfigPtr</i>	Pointer to structure of "afe_user_config_t".
----------------------	--

Returns

Execution status.

4.3.8.2 afe_status_t AFE_DRV_StructInitChnConfigDefault (afe_chn_config_t * chnConfigPtr)

This function fills the [afe_chn_config_t](#) structure with default settings. These setting are:

```
.hwTriggerEnable = false
.continuousConvEnable = false
.chnMode = kAfeNormal
.decimOSR = kAfeDecOsrOf64
.delay = 0
.pgaGainSel = kAfePgaGain1
.chnEvent = kAfeNoneReq
```

Parameters

<i>userConfigPtr</i>	Pointer to structure of "afe_chn_config_t".
----------------------	---

Returns

Execution status.

4.3.8.3 afe_status_t AFE_DRV_Init (uint32_t instance, afe_user_config_t * userConfigPtr)

This function configures the AFE module for the configuration which are shared by all channels.

Parameters

<i>instance</i>	The AFE instance number.
<i>userConfigPtr</i>	Pointer to structure of "afe_user_config_t". If startupCnt parameter is less than two, this value is calculated according to equation $Statup_cnt = (clk_freq/clk_div)*20e-6$.

Returns

Execution status.

4.3.8.4 afe_status_t AFE_DRV_ChnlInit (uint32_t instance, uint32_t chn, afe_chn_config_t * chnConfigPtr)

This function configures the selected AFE channel.

Parameters

<i>instance</i>	The AFE instance number.
-----------------	--------------------------

AFE Peripheral driver

<i>chn</i>	Channel which will be triggered.
<i>chnConfigPtr</i>	Pointer to structure of "afe_chn_config_t".

Returns

Execution status.

4.3.8.5 void AFE_DRV_Enable (uint32_t *instance*, bool *enable*)

This function enables or disables all channels whose SD_MOD_EN and DEC_EN bits are asserted.

Parameters

<i>instance</i>	The AFE instance number.
<i>enable</i>	Enable (true) or disable (false) all ADCs and filters.

4.3.8.6 void AFE_DRV_WaitConvDone (uint32_t *instance*, uint32_t *chn*)

This function holds the program until the last conversion is complete.

Parameters

<i>instance</i>	The AFE instance number.
<i>chn</i>	Channel which is triggered.

4.3.8.7 void AFE_DRV_WaitChnReady (uint32_t *instance*, uint32_t *chn*)

This function holds the program until the channel is ready for conversion.

Parameters

<i>instance</i>	The AFE instance number.
<i>chn</i>	Channel which is triggered.

4.3.8.8 void AFE_DRV_SoftTriggerConv (uint32_t *instance*, uint32_t *chnMask*)

This function triggers the AFE conversion by executing a software command. It starts the conversion on selected channels if the software trigger option is selected for the channels.

Parameters

<i>instance</i>	The AFE instance number.
<i>chn</i>	Channel(s) mask which is triggered.

4.3.8.9 bool AFE_DRV_GetChnFlag (uint32_t *instance*, uint32_t *chn*, afe_flag_t *flag*)

This function returns the selected flag of the desired channel.

Parameters

<i>instance</i>	The AFE instance number.
<i>chn</i>	The AFE Channel.
<i>flag</i>	Indicated event, see to "afe_flag_t".

Returns

Assertion of indicated event.

4.3.8.10 uint32_t AFE_DRV_GetChnConvValRaw (uint32_t *instance*, uint32_t *chn*)

This function returns the conversion value of the selected channel. The returned value could be left or right adjusted according to the AFE module configuration.

Parameters

<i>instance</i>	The AFE instance number.
<i>chn</i>	The AFE Channel.

Returns

Conversion value.

4.3.8.11 int32_t AFE_DRV_GetChnConvVal (uint32_t *instance*, uint32_t *chn*)

This function returns the conversion value of the selected channel. The returned value is in the two's complement (right adjusted) format.

AFE Peripheral driver

Parameters

<i>instance</i>	The AFE instance number.
<i>chn</i>	The AFE Channel.

Returns

Conversion value in 2's complement format.

4.3.8.12 void AFE_DRV_Deinit (uint32_t *instance*)

When the AFE is no longer used, calling this function shuts down the device to reduce power consumption.

Parameters

<i>instance</i>	The AFE instance number.
-----------------	--------------------------

4.3.8.13 void AFE_DRV_ChnDeinit (uint32_t *instance*, uint32_t *chn*)

De-initializes the selected AFE channel configuration and interrupt.

Parameters

<i>instance</i>	The AFE instance number.
<i>chn</i>	The AFE Channel.

4.3.8.14 void AFE_DRV_AssertDelayOk (uint32_t *instance*)

This function should be called after all desired channel's delay registers are loaded. Values in channel's delay registers are active after calling this function and after the conversation starts.

Parameters

<i>instance</i>	The AFE instance number.
-----------------	--------------------------

4.3.8.15 void AFE_DRV_SetPhaseDelays (uint32_t *instance*, afe_delay_config_t * *delayConfigPtr*)

This function sets the phase delays for channels. This delay is inserted before the trigger response of the decimation filters. The delay is used to provide a phase compensation between AFE channels in

step of prescaled modulator clock periods. The DelayOk bit is asserted in this function so the 'AFE_DRV_AssertDelayOk()' function doesn't have to be called. The delays for each channel are stored in a 'afe_delay_config_t' structure.

Parameters

<i>instance</i>	The AFE instance number.
<i>delayConfigPtr</i>	Pointer to structure of "afe_delay_config_t".

4.3.9 Variable Documentation

4.3.9.1 AFE_Type* const g_afeBase[]

4.3.9.2 const IRQn_Type g_afeIrqId[]

Chapter 5

Crossbar AND/OR/INVERT (AOI) Module

5.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Crossbar AND/OR/INVERT module of Kinetis devices.

Modules

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

AOI HAL Driver

5.2 AOI HAL Driver

5.2.1 Overview

The section describes the programming interface of the AOI HAL driver. AOI HAL driver is a set of API functions used to access and configure the AOI hardware registers.

Enumerations

- enum `aoi_status_t` {
`kStatus_AOI_Success` = 0U,
`kStatus_AOI_InvalidArgument` = 1U,
`kStatus_AOI_Failed` = 2U }
AOI status return codes.
- enum `aoi_input_config_t` {
`kAoiConfigLogicZero` = 0x0U,
`kAoiConfigInputSignal` = 0x1U,
`kAoiConfigInvInputSignal` = 0x2U,
`kAoiConfigLogicOne` = 0x3U }
- enum `aoi_product_term_t` {
`kAoiTerm0` = 0x0U,
`kAoiTerm1` = 0x1U,
`kAoiTerm2` = 0x2U,
`kAoiTerm3` = 0x3U }
Defines the product term numbers.
- enum `aoi_input_signal_index_t` {
`kAoiInputA` = 0x0U,
`kAoiInputB` = 0x1U,
`kAoiInputC` = 0x2U,
`kAoiInputD` = 0x3U }
AOI input signal indexes.
- enum `aoi_event_index_t` {
`kAoiEvent0` = 0x0U,
`kAoiEvent1` = 0x1U,
`kAoiEvent2` = 0x2U,
`kAoiEvent3` = 0x3U }
AOI event indexes, where an event is the collection of the four product terms (0, 1, 2, and 3) and the four signal inputs (A, B, C, and D).

Functions

- void `AOI_HAL_Init` (AOI_Type *base)
Initializes the AOI module to the reset state.
- void `AOI_HAL_Reset` (AOI_Type *base, `aoi_event_index_t` event)
Resets the configuration registers of a specific AOI event.

- void [AOI_HAL_SetSignalLogicUnit](#) (AOI_Type *base, [aoi_event_index_t](#) event, [aoi_product_term_t](#) productTerm, [aoi_input_signal_index_t](#) input, [aoi_input_config_t](#) config)
Defines the Boolean evaluation associated with the selected input in the selected product term of the desired event.
- [aoi_input_config_t](#) [AOI_HAL_GetSignalLogicUnit](#) (AOI_Type *base, [aoi_event_index_t](#) event, [aoi_product_term_t](#) productTerm, [aoi_input_signal_index_t](#) input)
Gets the Boolean evaluation associated with the selected input in the selected product term of the desired event.

5.2.2 Enumeration Type Documentation

5.2.2.1 enum aoi_status_t

Enumerator

- kStatus_AOI_Success* Success.
- kStatus_AOI_InvalidArgument* Invalid argument existed.
- kStatus_AOI_Failed* Execution failed.

5.2.2.2 enum aoi_input_config_t

Enumerator

- kAoiConfigLogicZero* Forces the input to logical zero.
- kAoiConfigInputSignal* Passes the input signal.
- kAoiConfigInvInputSignal* Inverts the input signal.
- kAoiConfigLogicOne* Forces the input to logical one.

5.2.2.3 enum aoi_product_term_t

Enumerator

- kAoiTerm0* Product term 0.
- kAoiTerm1* Product term 1.
- kAoiTerm2* Product term 2.
- kAoiTerm3* Product term 3.

5.2.2.4 enum aoi_input_signal_index_t

Enumerator

- kAoiInputA* Input configuration A.
- kAoiInputB* Input configuration B.

AOI HAL Driver

kAoiInputC Input configuration C.

kAoiInputD Input configuration D.

5.2.2.5 enum aoi_event_index_t

Enumerator

kAoiEvent0 Event 0 index.

kAoiEvent1 Event 1 index.

kAoiEvent2 Event 2 index.

kAoiEvent3 Event 3 index.

5.2.3 Function Documentation

5.2.3.1 void AOI_HAL_Init (AOI_Type * *base*)

This function initializes the module to the reset state. This state is defined in the chip Reference Manual, which is the power on reset value.

Parameters

<i>base</i>	Register base address for AOI module.
-------------	---------------------------------------

5.2.3.2 void AOI_HAL_Reset (AOI_Type * *base*, aoi_event_index_t *event*)

This function resets all product term inputs of a selected event to the reset values. This state is defined in the chip Reference Manual, which is the power on reset value.

Parameters

<i>base</i>	Register base address for AOI module.
<i>event</i>	Event of AOI to be reset of type aoi_event_index_t.

5.2.3.3 void AOI_HAL_SetSignalLogicUnit (AOI_Type * *base*, aoi_event_index_t *event*, aoi_product_term_t *productTerm*, aoi_input_signal_index_t *input*, aoi_input_config_t *config*)

This function defines the Boolean evaluation associated with the selected input in the selected product term of the desired event.

Parameters

<i>base</i>	Register base address for AOI module.
<i>event</i>	Number of the event which will be set of type <code>aoi_event_index_t</code> .
<i>productTerm</i>	The term which will be set of type <code>aoi_product_term_t</code> .
<i>input</i>	The input which will be set of type <code>aoi_input_signal_index_t</code> .
<i>config</i>	Selected input configuration of type <code>aoi_input_config_t</code> .

5.2.3.4 `aoi_input_config_t` AOI_HAL_GetSignalLogicUnit (AOI_Type * *base*, *aoi_event_index_t event*, *aoi_product_term_t productTerm*, *aoi_input_signal_index_t input*)

This function returns the Boolean evaluation associated with the selected input in the selected product term of the desired event.

Parameters

<i>base</i>	Register base address for AOI module.
<i>event</i>	Number of the event which will be set of type <code>aoi_event_index_t</code> .
<i>productTerm</i>	The product term which will be set of type <code>aoi_product_term_t</code> .
<i>input</i>	The input which will be set of type <code>aoi_input_signal_index_t</code> .

Returns

Selected input configuration of type `aoi_input_config_t`.

AOI Peripheral Driver

5.3 AOI Peripheral Driver

5.3.1 Overview

The section describes the programming interface of the AOI Peripheral driver.

5.3.2 Overview

The AOI peripheral driver configures the AOI (AND/OR/INVERT) module and handles the initialization and configuration of the AOI module.

5.3.3 Initialization

To initialize the AOI module, call the [AOI_DRV_Init\(\)](#) function. After initialization, the AOI module is set to a reset state.

```
// Initialize AOI module.  
AOI_DRV_Init(instance);
```

5.3.4 Event output configuration

The AOI module provides a universal boolean function generator using a four-term sum of products expression with each product term containing true or complement values of the four selected event inputs (A, B, C, D). The AOI is a highly programmable module for creating combinational boolean outputs for use as hardware triggers. Each selected input term in each product term can be configured to produce a logical 0 or 1 or pass the true or complement of the selected event input. To configure the selected AOI module event, call the API of the [AOI_DRV_SetEventLogic\(\)](#) function. To configure only one product term logic, call the [AOI_DRV_SetProductTermLogic\(\)](#) function. The AOI module does not support any special modes of operation.

5.3.5 Call diagram

1. Call the [AOI_DRV_Init\(\)](#) function to initialize AOI to a known state.
2. Optionally, call the [AOI_DRV_SetEventLogic\(\)](#) function to configure the Boolean evaluation in all product terms in one event. Use the configuration structure, [aoi_event_config_t](#) and number of the event, when calling this function.
3. Optionally, call the [AOI_DRV_SetProductTermLogic\(\)](#) function to configure the Boolean evaluation associated with all inputs in one product and in one event. Use the configuration structure, [aoi_product_term_config_t](#) and the product term and number of the event when calling this function.
4. Finally, the AOI works properly.

If you want to stop the AOI module, call the void [AOI_DRV_Deinit\(\)](#) function. This clears the AOI module configuration and disables the clock gate.

This is an example to initialize and configure the AOI driver for a possible use case. Because the AOI module function is directly connected with an XBAR (Inter-peripheral crossbar) module, other peripheral (PIT, CMP and XBAR) drivers are used to show full functionality of AOI module.

```

/* aoi_test.c */

#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include "board.h"
#include "fsl_cmp_driver.h"
#include "fsl_device_registers.h"
#include "fsl_aoi_driver.h"
#include "fsl_pit_driver.h"
#include "fsl_xbar_driver.h"

#define DEMO_CMP0_INSTANCE    (0U)
#define DEMO_PIT0_INSTANCE    (0U)

/***** Defi-
nition

```

Data Structures

- struct [aoi_product_term_config_t](#)
AOI product term configuration structure. [More...](#)
- struct [aoi_event_config_t](#)
AOI event configuration structure. [More...](#)

Functions

- [aoi_status_t AOI_DRV_Init](#) (uint32_t instance)
Initializes AOI module.
- [aoi_status_t AOI_DRV_Deinit](#) (uint32_t instance)
De-initializes the AOI module.
- [aoi_status_t AOI_DRV_ConfigEventLogic](#) (uint32_t instance, [aoi_event_index_t](#) event, const [aoi_event_config_t](#) *eventConfigPtr)
Configures an AOI event.
- [aoi_status_t AOI_DRV_ConfigProductTermLogic](#) (uint32_t instance, [aoi_event_index_t](#) event, [aoi_product_term_t](#) productTerm, const [aoi_product_term_config_t](#) *productTermConfigPtr)
Configures an AOI module product term in a specific event.

Variables

- AOI_Type *const [g_aoiBase](#) []
Table of base addresses for AOI instances.

AOI Peripheral Driver

5.3.6 Data Structure Documentation

5.3.6.1 struct aoi_product_term_config_t

Data Fields

- [aoi_input_config_t PTAC](#)
PTx_AC configuration.
- [aoi_input_config_t PTBC](#)
PTx_BC configuration.
- [aoi_input_config_t PTCC](#)
PTx_CC configuration.
- [aoi_input_config_t PTDC](#)
PTx_DC configuration.

5.3.6.1.0.6 Field Documentation

5.3.6.1.0.6.1 [aoi_input_config_t aoi_product_term_config_t::PTAC](#)

5.3.6.1.0.6.2 [aoi_input_config_t aoi_product_term_config_t::PTBC](#)

5.3.6.1.0.6.3 [aoi_input_config_t aoi_product_term_config_t::PTCC](#)

5.3.6.1.0.6.4 [aoi_input_config_t aoi_product_term_config_t::PTDC](#)

5.3.6.2 struct aoi_event_config_t

Defines structure AoiEventConfig and use the [AOI_DRV_ConfigEventLogic\(\)](#) function to make whole event configuration.

Data Fields

- [aoi_input_config_t PT1DC](#)
PT1_DC configuration.
- [aoi_input_config_t PT1CC](#)
PT1_CC configuration.
- [aoi_input_config_t PT1BC](#)
PT1_BC configuration.
- [aoi_input_config_t PT1AC](#)
PT1_AC configuration.
- [aoi_input_config_t PT0DC](#)
PT0_DC configuration.
- [aoi_input_config_t PT0CC](#)
PT0_CC configuration.
- [aoi_input_config_t PT0BC](#)
PT0_BC configuration.
- [aoi_input_config_t PT0AC](#)
PT0_AC configuration.
- [aoi_input_config_t PT3DC](#)

- PT3_DC configuration.*
- [aoi_input_config_t PT3CC](#)
PT3_CC configuration.
- [aoi_input_config_t PT3BC](#)
PT3_BC configuration.
- [aoi_input_config_t PT3AC](#)
PT3_AC configuration.
- [aoi_input_config_t PT2DC](#)
PT2_DC configuration.
- [aoi_input_config_t PT2CC](#)
PT2_CC configuration.
- [aoi_input_config_t PT2BC](#)
PT2_BC configuration.
- [aoi_input_config_t PT2AC](#)
PT2_AC configuration.

AOI Peripheral Driver

5.3.6.2.0.7 Field Documentation

5.3.6.2.0.7.1 `aoi_input_config_t aoi_event_config_t::PT1DC`

5.3.6.2.0.7.2 `aoi_input_config_t aoi_event_config_t::PT1CC`

5.3.6.2.0.7.3 `aoi_input_config_t aoi_event_config_t::PT1BC`

5.3.6.2.0.7.4 `aoi_input_config_t aoi_event_config_t::PT1AC`

5.3.6.2.0.7.5 `aoi_input_config_t aoi_event_config_t::PT0DC`

5.3.6.2.0.7.6 `aoi_input_config_t aoi_event_config_t::PT0CC`

5.3.6.2.0.7.7 `aoi_input_config_t aoi_event_config_t::PT0BC`

5.3.6.2.0.7.8 `aoi_input_config_t aoi_event_config_t::PT0AC`

5.3.6.2.0.7.9 `aoi_input_config_t aoi_event_config_t::PT3DC`

5.3.6.2.0.7.10 `aoi_input_config_t aoi_event_config_t::PT3CC`

5.3.6.2.0.7.11 `aoi_input_config_t aoi_event_config_t::PT3BC`

5.3.6.2.0.7.12 `aoi_input_config_t aoi_event_config_t::PT3AC`

5.3.6.2.0.7.13 `aoi_input_config_t aoi_event_config_t::PT2DC`

5.3.6.2.0.7.14 `aoi_input_config_t aoi_event_config_t::PT2CC`

5.3.6.2.0.7.15 `aoi_input_config_t aoi_event_config_t::PT2BC`

5.3.6.2.0.7.16 `aoi_input_config_t aoi_event_config_t::PT2AC`

5.3.7 Function Documentation

5.3.7.1 `aoi_status_t AOI_DRV_Init (uint32_t instance)`

This function initializes AOI module. It configures all the AOI module inputs of all events to the reset state (`kAoiLogicZero`).

This is an example to initialize the AOI module:

```
status = AOI_DRV_Init();
switch (status)
{
    //...
}
```

Parameters

<i>instance</i>	The instance number of the AOI peripheral
-----------------	---

Returns

kStatus_AOI_Success indicating successful initialization

5.3.7.2 aoi_status_t AOI_DRV_Deinit (uint32_t instance)

This function clears all configurations and shuts down the AOI module clock to reduce the power consumption.

Parameters

<i>instance</i>	The instance number of the AOI peripheral
-----------------	---

Returns

kStatus_AOI_Success indicating successful de-initialization

5.3.7.3 aoi_status_t AOI_DRV_ConfigEventLogic (uint32_t instance, aoi_event_index_t event, const aoi_event_config_t * eventConfigPtr)

This function configures an AOI event according to the aoiEventConfig structure. This function configures all inputs (A, B, C, and D) of all product terms (0, 1, 2, and 3) of a desired event. This is an example to set up the AOI Event structure:

```

aoi_event_config_t aoiEventConfig;

aoiEventConfig.PT0AC = kAoiConfigInputSignal;
aoiEventConfig.PT0BC = kAoiConfigLogicZero;
aoiEventConfig.PT0CC = kAoiConfigLogicZero;
aoiEventConfig.PT0DC = kAoiConfigLogicZero;

aoiEventConfig.PT1AC = kAoiConfigInvInputSignal;
aoiEventConfig.PT1BC = kAoiConfigLogicZero;
aoiEventConfig.PT1CC = kAoiConfigLogicZero;
aoiEventConfig.PT1DC = kAoiConfigInputSignal;

aoiEventConfig.PT2AC = kAoiConfigInputSignal;
aoiEventConfig.PT2BC = kAoiConfigInputSignal;
aoiEventConfig.PT2CC = kAoiConfigInputSignal;
aoiEventConfig.PT2DC = kAoiConfigLogicOne;

aoiEventConfig.PT3AC = kAoiConfigLogicOne;
aoiEventConfig.PT3BC = kAoiConfigLogicOne;
aoiEventConfig.PT3CC = kAoiConfigLogicOne;
aoiEventConfig.PT3DC = kAoiConfigLogicOne;

```

AOI Peripheral Driver

```

aoi_status_t status;
// In the function call below, the value of "2" indicates event #2 is being used.
status = AOI_DRV_ConfigEventLogic(0, 2, &aoiEventConfig);
switch (status)
{
    //...
}

```

Parameters

<i>instance</i>	The instance number of the AOI peripheral
<i>event</i>	Event which will be configured of type <code>aoi_event_index_t</code> .
<i>eventConfigPtr</i>	Pointer to type <code>aoi_event_config_t</code> structure. The user is responsible to fill out the members of this structure and pass the pointer to this function.

Returns

An error code or `kStatus_AOI_Success`.

5.3.7.4 `aoi_status_t AOI_DRV_ConfigProductTermLogic (uint32_t instance, aoi_event_index_t event, aoi_product_term_t productTerm, const aoi_product_term_config_t * productTermConfigPtr)`

This function configures an AOI module product terms for a specific event. The user has to select the event and the product term which is configured and fill the `AoiProductTermConfig` configuration structure.

Example:

```

aoi_product_term_config_t productTermConfigStruct;
aoi_status_t status;

productTermConfigStruct.PTAC = kAoiConfigLogicZero;
productTermConfigStruct.PTBC = kAoiConfigInputSignal;
productTermConfigStruct.PTCC = kAoiConfigInvInputSignal;
productTermConfigStruct.PTDC = kAoiConfigLogicOne;

// Configure product term 1 of event 3
status = AOI_DRV_ConfigProductTermLogic(0, 3,
    kAoiTerm1, &productTermConfigStruct);
switch (status)
{
    //...
}

```

Parameters

<i>instance</i>	The instance number of the AOI peripheral
<i>event</i>	Event which will be configured of type <code>aoi_event_index_t</code> .
<i>productTerm</i>	Product term which will be configured of type <code>aoi_product_term_t</code> .
<i>productTerm-ConfigPtr</i>	Pointer to type <code>aoi_product_term_config_t</code> structure. The user is responsible to fill out the members of this structure and pass the pointer to this function.

Returns

An error code or `kStatus_AOI_Success`.

5.3.8 Variable Documentation

5.3.8.1 `AOI_Type* const g_aoiBase[]`



Chapter 6 Comparator (CMP)

6.1 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](#)

CMP HAL Driver

6.2 CMP HAL Driver

6.2.1 Overview

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

Data Structures

- struct `cmp_comparator_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 the CMP module. [More...](#)

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 }`
Defines the selections of the 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 }`
Defines the selections of the filter sample counter.
- enum `cmp_dac_ref_volt_src_mode_t` {
`kCmpDacRefVoltSrcOf1 = 0U,`
`kCmpDacRefVoltSrcOf2 = 1U }`
Defines the selections of reference voltage source for the 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 the CMP channel mux.

- enum `cmp_sample_filter_mode_t` {


```

kCmpContinuousMode = 0U,
kCmpSampleWithNoFilteredMode = 1U,
kCmpSampleWithFilteredMode = 2U,
kCmpWindowedMode = 3U,
kCmpWindowedFilteredMode = 4U }

```

Definition selections of the sample and filter modes in the CMP module.

Functions

- void `CMP_HAL_Init` (CMP_Type *base)

Resets the CMP registers to a known state.
- void `CMP_HAL_ConfigComparator` (CMP_Type *base, const `cmp_comparator_config_t` *configPtr)

Configures the CMP comparator function.
- void `CMP_HAL_ConfigDacChn` (CMP_Type *base, const `cmp_dac_config_t` *configPtr)

Configures the CMP DAC function.
- void `CMP_HAL_ConfigSampleFilter` (CMP_Type *base, const `cmp_sample_filter_config_t` *configPtr)

Configures the CMP sample or the filter function.
- static void `CMP_HAL_Enable` (CMP_Type *base)

Enables the comparator in the CMP module.
- static void `CMP_HAL_Disable` (CMP_Type *base)

Disables the comparator in the CMP module.
- static bool `CMP_HAL_GetOutputLogic` (CMP_Type *base)

Gets the comparator logic output in the CMP module.
- static bool `CMP_HAL_GetOutputFallingFlag` (CMP_Type *base)

Gets the logic output falling edge event in the CMP module.
- static void `CMP_HAL_ClearOutputFallingFlag` (CMP_Type *base)

Clears the logic output falling edge event in the CMP module.
- static bool `CMP_HAL_GetOutputRisingFlag` (CMP_Type *base)

Gets the logic output rising edge event in the CMP module.
- static void `CMP_HAL_ClearOutputRisingFlag` (CMP_Type *base)

Clears the logic output rising edge event in the CMP module.

CMP HAL Driver

6.2.2 Data Structure Documentation

6.2.2.1 struct cmp_comparator_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_hystersis_mode_t hystersisMode](#)
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.

6.2.2.2 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.

6.2.2.2.0.8 Field Documentation

6.2.2.2.0.8.1 `cmp_filter_counter_mode_t` `cmp_sample_filter_config_t::filterCount`

See "`cmp_filter_counter_mode_t`".

6.2.2.3 `struct cmp_dac_config_t`

This structure holds the configuration for the 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

- bool `dacEnable`
Enable the internal 6-bit DAC.
- `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.

6.2.3 Enumeration Type Documentation

6.2.3.1 `enum cmp_status_t`

Enumerator

- `kStatus_CMP_Success` Success.
- `kStatus_CMP_InvalidArgument` Invalid argument existed.
- `kStatus_CMP_Failed` Execution failed.

6.2.3.2 `enum cmp_hysteresis_mode_t`

The hysteresis control level indicates the smallest window between the two inputs when asserting the change of output. See the chip Data Sheet for detailed electrical characteristics. Generally, the lower level represents the smaller window.

Enumerator

- `kCmpHysteresisOfLevel0` Level 0.
- `kCmpHysteresisOfLevel1` Level 1.
- `kCmpHysteresisOfLevel2` Level 2.
- `kCmpHysteresisOfLevel3` Level 3.

CMP HAL Driver

6.2.3.3 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

<i>kCmpFilterCountSampleOf0</i>	Disable the filter.
<i>kCmpFilterCountSampleOf1</i>	One sample must agree.
<i>kCmpFilterCountSampleOf2</i>	2 consecutive samples must agree
<i>kCmpFilterCountSampleOf3</i>	3 consecutive samples must agree
<i>kCmpFilterCountSampleOf4</i>	4 consecutive samples must agree
<i>kCmpFilterCountSampleOf5</i>	5 consecutive samples must agree
<i>kCmpFilterCountSampleOf6</i>	6 consecutive samples must agree
<i>kCmpFilterCountSampleOf7</i>	7 consecutive samples must agree

6.2.3.4 enum cmp_dac_ref_volt_src_mode_t

Enumerator

<i>kCmpDacRefVoltSrcOf1</i>	Vin1 - Vref_out.
<i>kCmpDacRefVoltSrcOf2</i>	Vin2 - Vdd.

6.2.3.5 enum cmp_chn_mux_mode_t

Enumerator

<i>kCmpInputChn0</i>	Comparator input channel 0.
<i>kCmpInputChn1</i>	Comparator input channel 1.
<i>kCmpInputChn2</i>	Comparator input channel 2.
<i>kCmpInputChn3</i>	Comparator input channel 3.
<i>kCmpInputChn4</i>	Comparator input channel 4.
<i>kCmpInputChn5</i>	Comparator input channel 5.
<i>kCmpInputChn6</i>	Comparator input channel 6.
<i>kCmpInputChn7</i>	Comparator input channel 7.
<i>kCmpInputChnDac</i>	Comparator input channel 7.

6.2.3.6 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

- $((\text{filter counter} * \text{filter prescaler}) + 1)$ bus clock for the filter function.

Enumerator

- kCmpContinuousMode* Continuous Mode.
- kCmpSampleWithNoFilteredMode* Sample, Non-Filtered Mode.
- kCmpSampleWithFilteredMode* Sample, Filtered Mode.
- kCmpWindowedMode* Window Mode.
- kCmpWindowedFilteredMode* Window/Filtered Mode.

6.2.4 Function Documentation

6.2.4.1 void CMP_HAL_Init (CMP_Type * base)

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

Parameters

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

6.2.4.2 void CMP_HAL_ConfigComparator (CMP_Type * base, const cmp_comparator_config_t * configPtr)

This function configures the CMP comparator function.

Parameters

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

CMP HAL Driver

<i>configPtr</i>	Pointer to configuration structure. See to "cmp_comparator_config_t".
------------------	---

6.2.4.3 void CMP_HAL_ConfigDacChn (CMP_Type * *base*, const cmp_dac_config_t * *configPtr*)

This function configures the CMP DAC function.

Parameters

<i>base</i>	Register base address for the module.
<i>configPtr</i>	Pointer to configuration structure. See to "cmp_dac_config_t".

6.2.4.4 void CMP_HAL_ConfigSampleFilter (CMP_Type * *base*, const cmp_sample_filter_config_t * *configPtr*)

This function configures the CMP sample or filter function.

Parameters

<i>base</i>	Register base address for the module.
<i>configPtr</i>	Pointer to configuration structure. See to "cmp_sample_filter_config_t".

6.2.4.5 static void CMP_HAL_Enable (CMP_Type * *base*) [inline], [static]

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

Parameters

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

6.2.4.6 static void CMP_HAL_Disable (CMP_Type * *base*) [inline], [static]

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

Parameters

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

6.2.4.7 static bool CMP_HAL_GetOutputLogic (CMP_Type * *base*) [inline], [static]

This function gets the comparator logic output in the CMP module. It returns the current value of the analog comparator output. The value is reset to 0 and read as de-asserted value when the CMP module is disabled. When setting to invert mode, the comparator logic output is inverted as well.

Parameters

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

Returns

The logic output is assert or not.

6.2.4.8 static bool CMP_HAL_GetOutputFallingFlag (CMP_Type * *base*) [inline], [static]

This function gets the logic output falling edge event in the CMP module. It detects a falling-edge on COUT and returns the asserted state when the falling-edge on COUT has occurred.

Parameters

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

Returns

The falling-edge on COUT has occurred or not.

6.2.4.9 static void CMP_HAL_ClearOutputFallingFlag (CMP_Type * *base*) [inline], [static]

This function clears the logic output falling edge event in the CMP module.

CMP HAL Driver

Parameters

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

6.2.4.10 **static bool CMP_HAL_GetOutputRisingFlag (CMP_Type * *base*) [inline], [static]**

This function gets the logic output rising edge event in the CMP module. It detects a rising-edge on COUT and returns the asserted state when the rising-edge on COUT has occurred.

Parameters

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

Returns

The rising-edge on COUT has occurred or not.

6.2.4.11 **static void CMP_HAL_ClearOutputRisingFlag (CMP_Type * *base*) [inline], [static]**

This function clears the logic output rising edge event in the CMP module.

Parameters

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

6.3 CMP Peripheral Driver

6.3.1 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.

6.3.2 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.

6.3.3 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 hold 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 to 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 Peripheral Driver

```

// cmp_test.c //

#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) );

int main(void)
{
    cmp_comparator_config_t testCmpUserConfigStruct;
    cmp_sample_filter_config_t testCmpSampleFilterConfigStruct;
    cmp_dac_config_t testCmpDacConfigStruct;

    // Init IO for CMP0_IN0. Board don't have pull up resistor, so internal resistor need to be enabled//
#ifdef TWR_K64F120M
    PORT_HAL_SetPullMode(PORTC, 6U, kPortPullUp);
    PORT_HAL_SetPullCmd(PORTC, 6U, true);
#else
#endif

    PRINTF("CMP PD TEST: Start...\r\n");
    // 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, &testCmpStateStruct, &testCmpUserConfigStruct);

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

    // Configure the Sample/Filter Mode. //
    testCmpSampleFilterConfigStruct.workMode = kCmpContinuousMode;
    testCmpSampleFilterConfigStruct.useExtSampleOrWindow = false;
    testCmpSampleFilterConfigStruct.filterClkDiv = 16U;
    testCmpSampleFilterConfigStruct.filterCount =
        kCmpFilterCountSampleOf4;

    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("^ CMP output rising event occur!\r\n");
        PRINTF(" CMP output level %d\r\n", CMP_DRV_GetOutputLogic(
TEST_CMP_INSTANCE) );
        bRisingEvent = false;
    }
    if (bFallingEvent)
    {
        PRINTF("v CMP output falling event occur!\r\n");
        PRINTF(" CMP output level %d\r\n", CMP_DRV_GetOutputLogic(
TEST_CMP_INSTANCE) );
        bFallingEvent = false;
    }
}
CMP_DRV_Deinit (TEST_CMP_INSTANCE);
PRINTF("CMP PD Test ");
PRINTF("Succeed\r\n");
while(1){}
}

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_state_t`
Internal driver state information. *More...*

Enumerations

- enum `cmp_flag_t` {
`kCmpFlagOfCoutRising = 0U,`
`kCmpFlagOfCoutFalling = 1U }`
Defines type of flags for the CMP event.

CMP Peripheral Driver

Functions

- `cmp_status_t CMP_DRV_StructInitUserConfigDefault (cmp_comparator_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_state_t *userStatePtr, const cmp_comparator_config_t *userConfigPtr)`
Initializes the CMP module.
- `cmp_status_t 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_ConfigDacChn (uint32_t instance, const cmp_dac_config_t *dacConfigPtr)`
Enables the internal DAC in the CMP module.
- `cmp_status_t CMP_DRV_ConfigSampleFilter (uint32_t instance, const cmp_sample_filter_config_t *configPtr)`
Configures the Sample feature in the CMP module.
- `bool CMP_DRV_GetOutputLogic (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

- `CMP_Type *const g_cmpBase []`
Table of base addresses for CMP instances.
- `const IRQn_Type g_cmpIrqId [CMP_INSTANCE_COUNT]`
Table to save CMP IRQ enumeration numbers defined in CMSIS header file.

6.3.4 Data Structure Documentation

6.3.4.1 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.

6.3.5 Enumeration Type Documentation

6.3.5.1 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.

6.3.6 Function Documentation

6.3.6.1 cmp_status_t CMP_DRV_StructInitUserConfigDefault (cmp_comparator_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 :

- .hysteresisMode = kCmpHysteresisOfLevel0
- .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".

CMP Peripheral Driver

Returns

Execution status.

6.3.6.2 `cmp_status_t CMP_DRV_Init (uint32_t instance, cmp_state_t * userStatePtr,
const cmp_comparator_config_t * userConfigPtr)`

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

Parameters

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

Returns

Execution status.

6.3.6.3 cmp_status_t 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.
-----------------	------------------

Returns

Execution status.

6.3.6.4 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.
-----------------	------------------

6.3.6.5 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.

CMP Peripheral Driver

Parameters

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

6.3.6.6 `cmp_status_t CMP_DRV_ConfigDacChn (uint32_t instance, const 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.

Parameters

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

Returns

Execution status.

6.3.6.7 `cmp_status_t CMP_DRV_ConfigSampleFilter (uint32_t instance, const 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.

6.3.6.8 `bool CMP_DRV_GetOutputLogic (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.

6.3.6.9 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.

6.3.6.10 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".

6.3.7 Variable Documentation

6.3.7.1 CMP_Type* const g_cmpBase[]

6.3.7.2 const IRQn_Type g_cmplrqld[CMP_INSTANCE_COUNT]



Chapter 7

Computer Operating Properly (COP) Watchdog Timer

7.1 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 Peripheral Driver](#)

COP HAL driver

7.2 COP HAL driver

7.2.1 Overview

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

Data Structures

- struct `cop_config_t`
Data structure to initialize the COP. [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.
- enum `cop_status_t` {
`kStatus_COP_Success` = 0x0U,
`kStatus_COP_Fail` = 0x01,
`kStatus_COP_NotInitialized` = 0x2U,
`kStatus_COP_NullArgument` = 0x3U }
cop status return codes.

COP HAL.

- void `COP_HAL_SetConfig` (SIM_Type *base, const `cop_config_t` *configPtr)
Configures the COP Watchdog.
- static void `COP_HAL_Enable` (void)
Enables the COP Watchdog.
- static void `COP_HAL_Disable` (SIM_Type *base)
Disables the COP Watchdog.
- static bool `COP_HAL_IsEnable` (SIM_Type *base)
Determines whether the COP is enabled.
- static void `COP_HAL_Refresh` (SIM_Type *base)
Servicing the COP Watchdog.
- static void `COP_HAL_ResetSystem` (SIM_Type *base)
Resets the system.
- void `COP_HAL_Init` (SIM_Type *base)
Restores the COP module to the reset value.

7.2.2 Data Structure Documentation

7.2.2.1 struct cop_config_t

This structure is used to initialize the COP during the cop_init function call. It contains all COP configurations.

Data Fields

- bool [copWindowModeEnable](#)
Set COP watchdog run mode—Window mode or Normal mode.
- [cop_clock_source_t](#) [copClockSource](#)
Set COP watchdog clock source.
- [cop_timeout_cycles_t](#) [copTimeout](#)
Set COP watchdog timeout value.

7.2.3 Enumeration Type Documentation

7.2.3.1 enum cop_clock_source_t

Enumerator

kCopLpoClock LPO clock, 1K HZ.
kCopBusClock BUS clock.

7.2.3.2 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

7.2.3.3 enum cop_status_t

Enumerator

kStatus_COP_Success COP operation Succeed.
kStatus_COP_Fail COP operation Failed.
kStatus_COP_NotInitialized COP is not initialized yet.
kStatus_COP_NullArgument Argument is NULL.

COP HAL driver

7.2.4 Function Documentation

7.2.4.1 void COP_HAL_SetConfig (SIM_Type * *base*, const cop_config_t * *configPtr*)

The COP control register is write once after reset.

Parameters

<i>base</i>	The COP peripheral base address
<i>configPtr</i>	configure COP control register

7.2.4.2 static void COP_HAL_Enable (void) [inline], [static]

After reset the COP is enabled.

7.2.4.3 static void COP_HAL_Disable (SIM_Type * base) [inline], [static]

This function disables the COP Watchdog and should be called after reset if your application does not need the COP Watchdog.

Parameters

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

7.2.4.4 static bool COP_HAL_IsEnable (SIM_Type * base) [inline], [static]

This function checks whether the COP is running.

Parameters

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

Returns

State of the module

Return values

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

7.2.4.5 static void COP_HAL_Refresh (SIM_Type * base) [inline], [static]

This function resets the COP timeout by writing 0x55 then 0xAA. Writing any other value generates a system reset. The writing operations should be atomic.

COP HAL driver

Parameters

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

7.2.4.6 static void COP_HAL_ResetSystem (SIM_Type * *base*) [inline], [static]

This function resets the system.

Parameters

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

7.2.4.7 void COP_HAL_Init (SIM_Type * *base*)

This function restores the COP module to the reset value.

Parameters

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

7.3 COP Peripheral Driver

7.3.1 Overview

The section describes the programming interface of the COP Peripheral driver. The COP driver configures the COP and provides a simple way to initialize and configure the COP.

7.3.2 COP Initialization

To initialize the COP module, call the `COP_DRV_Init()` function and pass in the user configuration structure. This function automatically enables the COP module and clock.

After the `COP_DRV_Init()` function is called, the COP is enabled and its counter is working. Therefore, the `COP_DRV_Refresh()` function should be called before the COP times out.

This example code shows how to initialize and configure the driver:

```
// Define device configuration.
cop_user_config_t copInit =
{
    .copWindowModeEnable = (uint8_t)false,           //Disable COP window mode
#ifdef FSL_FEATURE_COP_HAS_LONGTIME_MODE
    .copTimeoutMode      = kCopShortTimeoutMode,     //configure COP timeout mode
#else
    .copClockSelect      = kCopLpoClock,             //configure COP clock
    source
#endif
    .copTimeout          = kCopTimeout_short_2to5_or_long_2to13, //
    configure COP timeout cycles
#ifdef FSL_FEATURE_COP_HAS_LONGTIME_MODE
    .copStopModeEnable   = (uint8_t)false,           //Disable COP in stop mode
    .copDebugModeEnable  = (uint8_t)false,           //Disable COP in debug mode
    .copClockSource      = kCopLpoClock,             //configure COP clock
    source
#endif
};

// Initialize COP.
COP_DRV_Init(instance, &init);
```

7.3.3 COP Refresh

After the COP is enabled, the `COP_DRV_Refresh()` function should be called periodically to prevent the COP from timing out.

Otherwise, a reset is asserted. This is called the "Feed Dog".

7.3.4 COP Reset Count

The COP can reset the system.

1. After `COP_DRV_Reset()`, the chip is be reset.

COP Peripheral Driver

Variables

- SIM_Type *const [g_copBase](#) []
Table of base addresses for COP instances.

COP Driver

- [cop_status_t COP_DRV_Init](#) (uint32_t instance, const [cop_config_t](#) *initPtr)
Initializes the COP.
- void [COP_DRV_Disable](#) (uint32_t instance)
Disables the COP Watchdog.
- void [COP_DRV_Refresh](#) (uint32_t instance)
Resets the COP timeout counter.
- bool [COP_DRV_IsRunning](#) (uint32_t instance)
Gets the COP running status.
- void [COP_DRV_ResetSystem](#) (uint32_t instance)
Resets the system.

7.3.5 Function Documentation

7.3.5.1 [cop_status_t COP_DRV_Init](#) (uint32_t instance, const cop_config_t * initPtr)

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

Parameters

<i>instance</i>	The COP peripheral instance number.
<i>initPtr</i>	COP Initialize data structure.

7.3.5.2 void [COP_DRV_Disable](#) (uint32_t instance)

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

Parameters

<i>instance</i>	The COP peripheral instance number.
-----------------	-------------------------------------

7.3.5.3 void COP_DRV_Refresh (uint32_t *instance*)

This function feeds the COP. It sets the COP timer count to zero and should be called before the COP timer expires. Otherwise, a RESET is asserted.

COP Peripheral Driver

Parameters

<i>instance</i>	The COP peripheral instance number.
-----------------	-------------------------------------

7.3.5.4 bool COP_DRV_IsRunning (uint32_t *instance*)

This function gets the COP running status.

Parameters

<i>instance</i>	The COP peripheral instance number.
-----------------	-------------------------------------

Returns

COP running status; 0 means not running; 1 means running

7.3.5.5 void COP_DRV_ResetSystem (uint32_t *instance*)

This function resets the device.

Parameters

<i>instance</i>	The COP peripheral instance number.
-----------------	-------------------------------------

7.3.6 Variable Documentation

7.3.6.1 SIM_Type* const g_copBase[]

Chapter 8

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

8.1 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](#)

CADC HAL Driver

8.2 CADC HAL Driver

8.2.1 Overview

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

Data Structures

- struct `cadc_controller_config_t`
Defines a structure to configure the CyclicADC module during initialization. [More...](#)
- struct `cadc_converter_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_status_t` {
`kStatus_CADC_Success = 0U,`
`kStatus_CADC_InvalidArgument = 1U,`
`kStatus_CADC_Failed = 2U }`
CADC status return codes.
- enum `cadc_diff_chn_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 to enumerate the zero crossing detection mode for each slot.

- enum `cadc_gain_mode_t` {
`kCAdcSGainBy1 = 0U,`
`kCAdcSGainBy2 = 1U,`
`kCAdcSGainBy4 = 2U }`

Defines the type to enumerate the 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 to enumerate the speed mode for each converter.

- enum `cadc_dma_trigger_src_t` {
`kCAdcDmaTriggeredByEndOfScan = 0U,`
`kCAdcDmaTriggeredByConvReady = 1U }`

Defines the type of DMA trigger source for each converter.

Functions

- void `CADC_HAL_Init` (ADC_Type *base)
Initializes all ADC registers to a known state.
- void `CADC_HAL_ConfigController` (ADC_Type *base, const `cadc_controller_config_t` *configPtr)
Configures the common features in cyclic ADC module.
- void `CADC_HAL_ConfigConvA` (ADC_Type *base, const `cadc_converter_config_t` *configPtr)
Configures the features for the converter A.
- void `CADC_HAL_ConfigConvB` (ADC_Type *base, const `cadc_converter_config_t` *configPtr)
Configures the features for the conversion B.
- void `CADC_HAL_ConfigChn` (ADC_Type *base, const `cadc_chn_config_t` *configPtr)
Configures the feature for the sample channel.
- void `CADC_HAL_ConfigSeqSlot` (ADC_Type *base, uint32_t slotIdx, const `cadc_slot_config_t` *configPtr)
Configures the features for the sample sequence slot.
- static void `CADC_HAL_SetConvAStartCmd` (ADC_Type *base)
Executes the command that starts conversion of the converter A.
- static void `CADC_HAL_SetConvBStartCmd` (ADC_Type *base)
Executes the command that start conversion of the converter B.
- static void `CADC_HAL_SetConvAPowerDownCmd` (ADC_Type *base, bool enable)
Shuts down the conversion power manually for the converter A.
- static void `CADC_HAL_SetConvBPowerDownCmd` (ADC_Type *base, bool enable)
Shuts down the conversion power manually for the converter B.
- static bool `CADC_HAL_GetConvAInProgressFlag` (ADC_Type *base)

CADC HAL Driver

- Gets the flag whether the converter A is in process.*

 - static bool `CADC_HAL_GetConvBInProgressFlag` (ADC_Type *base)
- Gets the flag whether the converter B is in process.*

 - static bool `CADC_HAL_GetConvAEndOfScanIntFlag` (ADC_Type *base)
- Gets the flag whether the converter A has finished the conversion.*

 - static bool `CADC_HAL_GetConvBEndOfScanIntFlag` (ADC_Type *base)
- Gets the flag whether the converter B has finished the conversion.*

 - static void `CADC_HAL_ClearConvAEndOfScanIntFlag` (ADC_Type *base)
- Clears the flag that finishes the conversion of the converter A.*

 - static void `CADC_HAL_ClearConvBEndOfScanIntFlag` (ADC_Type *base)
- Clears the flag that finishes the conversion of the converter B.*

 - static bool `CADC_HAL_GetZeroCrossingIntFlag` (ADC_Type *base)
- Gets the flag whether a sample zero-crossing event has happened.*

 - static uint16_t `CADC_HAL_GetSlotZeroCrossingFlag` (ADC_Type *base, uint16_t slotIdxMask)
- Gets the flag whether a sample zero-crossing event has happened.*

 - static void `CADC_HAL_ClearSlotZeroCrossingFlag` (ADC_Type *base, uint16_t slotIdxMask)
- Clears the flags of a sample zero-crossing events.*

 - static bool `CADC_HAL_GetLowLimitIntFlag` (ADC_Type *base)
- Gets the flag whether any sample value is lower than the low limit value.*

 - static uint16_t `CADC_HAL_GetSlotLowLimitFlag` (ADC_Type *base, uint16_t slotIdxMask)
- Gets the flags whether a sample value is lower than the low limit value.*

 - static void `CADC_HAL_ClearSlotLowLimitFlag` (ADC_Type *base, uint16_t slotIdxMask)
- Clears the flags of the sample low limit event.*

 - static bool `CADC_HAL_GetHighLimitIntFlag` (ADC_Type *base)
- Gets the flag whether any sample value is higher than the high limit value.*

 - static uint16_t `CADC_HAL_GetSlotHighLimitFlag` (ADC_Type *base, uint16_t slotIdxMask)
- Gets the flags whether a sample value is higher than the high limit value.*

 - static void `CADC_HAL_ClearSlotHighLimitFlag` (ADC_Type *base, uint16_t slotIdxMask)
- Clears the flags of the sample high limit event.*

 - static uint16_t `CADC_HAL_GetSlotReadyFlag` (ADC_Type *base, uint16_t slotIdxMask)
- Gets the flags whether a sample value is ready.*

 - static bool `CADC_HAL_GetConvAPowerDownFlag` (ADC_Type *base)
- Gets the flag whether the converter A is powered down.*

 - static bool `CADC_HAL_GetConvBPowerDownFlag` (ADC_Type *base)
- Gets the flag whether the converter B is powered down.*

 - static uint16_t `CADC_HAL_GetSampleValue` (ADC_Type *base, uint16_t slotIdx)
- Gets the sample value.*

8.2.2 Data Structure Documentation

8.2.2.1 struct `cadc_controller_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_t dmaSrc**
DMA trigger source.
- bool **autoStandbyEnable**
Auto standby mode enable.
- uint16_t **powerUpDelayCount**
Power up delay.
- bool **autoPowerDownEnable**
Auto power down mode enable.

8.2.2.2 struct **cadc_converter_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.
- uint16_t **clkDivValue**
ADC clock divider from the bus clock.
- 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.
- uint16_t **sampleWindowCount**
Sample window count.

8.2.2.3 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

CADC HAL Driver

when set to operate as a single end sample.

Data Fields

- `cadc_diff_chn_t diffChns`
Select the differential channel pair.
- `cadc_chn_sel_mode_t diffSelMode`
Select which channel is indicated in a pair.
- `cadc_gain_mode_t gainMode`
Gain mode for each channel.

8.2.2.4 struct `cadc_slot_config_t`

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

Data Fields

- bool `slotDisable`
Keep the slot unavailable.
- bool `syncPointEnable`
Sample waits for an enabled SYNC input to occur.
- bool `syncIntEnable`
Scan interrupt enable.
- `cadc_diff_chn_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.

8.2.3 Enumeration Type Documentation

8.2.3.1 enum `cadc_status_t`

Enumerator

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

8.2.3.2 enum cadc_diff_chn_t

Note, "cadc_diff_chn_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.

8.2.3.3 enum cadc_chn_sel_mode_t

Note, "cadc_diff_chn_t" and "cadc_chn_sel_mode_t" can determine selecting the single ADC sample channel.

Enumerator

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

8.2.3.4 enum cadc_scan_mode_t

See the ADC_CTRL1[SMODE] in the chip Reference Manual for detailed information about the ADC converter scan mode.

Enumerator

- kCAdcScanOnceSequential* Once (single) sequential.
- kCAdcScanOnceParallel* Once parallel.
- kCAdcScanLoopSequential* Loop sequential.
- kCAdcScanLoopParallel* Loop parallel.
- kCAdcScanTriggeredSequential* Triggered sequential.
- kCAdcScanTriggeredParalled* Triggered parallel (default).

8.2.3.5 enum cadc_zero_crossing_mode_t

Enumerator

- kCAdcZeroCrossingDisable* Zero crossing detection disabled.

CADC HAL Driver

kCAdcZeroCrossingAtRisingEdge Enable for positive to negative sign change.

kCAdcZeroCrossingAtFallingEdge Enable for negative to positive sign change.

kCAdcZeroCrossingAtBothEdge Enable for any sign change.

8.2.3.6 enum cadc_gain_mode_t

Enumerator

kCAdcSGainBy1 x1 amplification.

kCAdcSGainBy2 x2 amplification.

kCAdcSGainBy4 x4 amplification.

8.2.3.7 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.

8.2.3.8 enum cadc_dma_trigger_src_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

Enumerator

kCAdcDmaTriggeredByEndOfScan DMA trigger source is end of scan interrupt.

kCAdcDmaTriggeredByConvReady DMA trigger source is RDY status.

8.2.4 Function Documentation

8.2.4.1 void CADC_HAL_Init (ADC_Type * base)

The initial states of ADC registers are set as specified in the chip Reference Manual.

Parameters

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

8.2.4.2 void CADC_HAL_ConfigController (ADC_Type * *base*, const *cadc_controller_config_t* * *configPtr*)

This function configures the common features in cyclic ADC module. For detailed items, see the "cadc_controller_config_t".

Parameters

<i>base</i>	Register base address for the module.
<i>configPtr</i>	Pointer to configuration structure.

8.2.4.3 void CADC_HAL_ConfigConvA (ADC_Type * *base*, const *cadc_converter_config_t* * *configPtr*)

This function configures the features for the converter A. For detailed items, see the "cadc_converter_config_t".

Parameters

<i>base</i>	Register base address for the module.
<i>configPtr</i>	Pointer to configuration structure.

8.2.4.4 void CADC_HAL_ConfigConvB (ADC_Type * *base*, const *cadc_converter_config_t* * *configPtr*)

This function configures the features for the conversion B. For detailed items, see the "cadc_converter_config_t".

Parameters

<i>base</i>	Register base address for the module.
<i>configPtr</i>	Pointer to configuration structure.

CADC HAL Driver

8.2.4.5 void CADC_HAL_ConfigChn (ADC_Type * *base*, const cadc_chn_config_t * *configPtr*)

This function configures the features for the sample channel. For detailed items, see the "cadc_chn_config_t".

Parameters

<i>base</i>	Register base address for the module.
<i>configPtr</i>	Pointer to configuration structure.

8.2.4.6 void CADC_HAL_ConfigSeqSlot (ADC_Type * *base*, uint32_t *slotIdx*, const cadc_slot_config_t * *configPtr*)

This function configures the features for the sample sequence slot. For detailed items, see the "cadc_slot_config_t".

Parameters

<i>base</i>	Register base address for the module.
<i>slotIdx</i>	Sample slot index.
<i>configPtr</i>	Pointer to configuration structure.

8.2.4.7 static void CADC_HAL_SetConvAStartCmd (ADC_Type * *base*) [inline], [static]

This function executes the command that start the conversion of the converter A when using the software trigger.

Parameters

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

8.2.4.8 static void CADC_HAL_SetConvBStartCmd (ADC_Type * *base*) [inline], [static]

This function executes the command that start the conversion of the converter B when using the software trigger.

Parameters

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

CADC HAL Driver

8.2.4.9 `static void CADC_HAL_SetConvAPowerDownCmd (ADC_Type * base, bool enable) [inline], [static]`

This function shuts down the conversion power manually for the conversion A. The conversion stops immediately after calling this function.

Parameters

<i>base</i>	Register base address for the module.
<i>enable</i>	Switcher to enable the feature or not.

8.2.4.10 static void CADC_HAL_SetConvBPowerDownCmd (ADC_Type * *base*, bool *enable*) [inline], [static]

This function shuts down the conversion power manually for the conversion B. The conversion stops immediately after calling this function.

Parameters

<i>base</i>	Register base address for the module.
<i>enable</i>	Switcher to enable the feature or not.

8.2.4.11 static bool CADC_HAL_GetConvAInProgressFlag (ADC_Type * *base*) [inline], [static]

This function gets the flag whether the converter A is in process.

Parameters

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

Returns

The event is asserted or not.

8.2.4.12 static bool CADC_HAL_GetConvBInProgressFlag (ADC_Type * *base*) [inline], [static]

This function gets the flag whether the converter B is in process.

Parameters

CADC HAL Driver

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

Returns

The event is asserted or not.

8.2.4.13 **static bool CADC_HAL_GetConvAEndOfScanIntFlag (ADC_Type * *base*)** **[inline], [static]**

This function gets the flag whether the converter A has finished the conversion.

Parameters

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

Returns

The event is asserted or not.

8.2.4.14 **static bool CADC_HAL_GetConvBEndOfScanIntFlag (ADC_Type * *base*)** **[inline], [static]**

This function gets the flag whether the converter B has finished the conversion.

Parameters

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

Returns

The event is asserted or not.

8.2.4.15 **static void CADC_HAL_ClearConvAEndOfScanIntFlag (ADC_Type * *base*)** **[inline], [static]**

This function clears the flag that finishes the conversion of the converter A.

Parameters

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

**8.2.4.16 static void CADC_HAL_ClearConvBEndOfScanIntFlag (ADC_Type * *base*)
[inline], [static]**

This function clears the flag that finishes the conversion of the converter B.

Parameters

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

**8.2.4.17 static bool CADC_HAL_GetZeroCrossingIntFlag (ADC_Type * *base*)
[inline], [static]**

This function gets the flag whether any sample zero-crossing event has happened.

Parameters

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

Returns

The event is asserted or not.

**8.2.4.18 static uint16_t CADC_HAL_GetSlotZeroCrossingFlag (ADC_Type * *base*,
uint16_t *slotIdxMask*) [inline], [static]**

This function gets the flags whether a sample zero-crossing event has happened.

Parameters

<i>base</i>	Register base address for the module.
<i>slotIdxMask</i>	Mask for indicated slots.

Returns

flags whether the event are happened for indicated slots.

CADC HAL Driver

8.2.4.19 `static void CADC_HAL_ClearSlotZeroCrossingFlag (ADC_Type * base, uint16_t slotIdxMask) [inline], [static]`

This function clears the flags of the sample zero-crossing events.

Parameters

<i>base</i>	Register base address for the module.
<i>slotIdxMask</i>	Mask for indicated slots.

8.2.4.20 static bool CADC_HAL_GetLowLimitIntFlag (ADC_Type * *base*) [inline], [static]

This function gets the flag whether any sample value is lower than the low limit value.

Parameters

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

Returns

The event is asserted or not.

8.2.4.21 static uint16_t CADC_HAL_GetSlotLowLimitFlag (ADC_Type * *base*, uint16_t *slotIdxMask*) [inline], [static]

This function gets the flags whether a samples value is lower than the low limit value.

Parameters

<i>base</i>	Register base address for the module.
<i>slotIdxMask</i>	Mask for indicated slots.

Returns

flags whether the event are happened for indicated slots.

8.2.4.22 static void CADC_HAL_ClearSlotLowLimitFlag (ADC_Type * *base*, uint16_t *slotIdxMask*) [inline], [static]

This function clears the flags of the sample low limit event.

CADC HAL Driver

Parameters

<i>base</i>	Register base address for the module.
<i>slotIdxMask</i>	Mask for indicated slots.

8.2.4.23 **static bool CADC_HAL_GetHighLimitIntFlag (ADC_Type * *base*) [inline], [static]**

This function gets the flag whether any sample value is higher than the high limit value.

Parameters

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

Returns

The event is asserted or not.

8.2.4.24 **static uint16_t CADC_HAL_GetSlotHighLimitFlag (ADC_Type * *base*, uint16_t *slotIdxMask*) [inline], [static]**

This function gets the flags whether a sample value is higher than the high limit value.

Parameters

<i>base</i>	Register base address for the module.
<i>slotIdxMask</i>	Mask for indicated slots.

Returns

flags whether the event are happened for indicated slots.

8.2.4.25 **static void CADC_HAL_ClearSlotHighLimitFlag (ADC_Type * *base*, uint16_t *slotIdxMask*) [inline], [static]**

This function clears the flags of the sample high limit event.

Parameters

<i>base</i>	Register base address for the module.
<i>slotIdxMask</i>	Mask for indicated slots.

8.2.4.26 static uint16_t CADC_HAL_GetSlotReadyFlag (ADC_Type * *base*, uint16_t *slotIdxMask*) [inline], [static]

This function gets the flags whether a sample value is ready.

Parameters

<i>base</i>	Register base address for the module.
<i>slotIdxMask</i>	Mask for indicated slots.

Returns

flags whether the event are happened for indicated slots.

8.2.4.27 static bool CADC_HAL_GetConvAPowerDownFlag (ADC_Type * *base*) [inline], [static]

This function gets the flag whether the converter A is powered down.

Parameters

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

Returns

The event is asserted or not.

8.2.4.28 static bool CADC_HAL_GetConvBPowerDownFlag (ADC_Type * *base*) [inline], [static]

This function gets the flag whether the converter B is powered down.

CADC HAL Driver

Parameters

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

Returns

The event is asserted or not.

8.2.4.29 `static uint16_t CADC_HAL_GetSampleValue (ADC_Type * base, uint16_t slotIdx) [inline], [static]`

This function gets the sample value. Note that the 3 LSBs are not available. When the differential conversion is executed, the MSB is the sign bit.

Parameters

<i>base</i>	Register base address for the module.
<i>slotIdx</i>	Index of slot in conversion sequence.

Returns

sample value with sign bit in MSB.

8.3 CADC Peripheral Driver

8.3.1 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.

8.3.2 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.

8.3.3 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:

CADC Peripheral Driver

- 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.

8.3.4 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.

8.3.5 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_controller_config_t g_AdcUserConfigStruct;
cadc_chn_config_t g_AdcChnConfigStruct;
cadc_converter_config_t g_AdcConvConfigStruct;
cadc_slot_config_t g_AdcSlotConfigStruct;

static uint32_t CADC_DRV_TEST_PollingMode(uint32_t instance);

int main(void)
{
    uint32_t idx;
    uint32_t errCounter = 0U;
    uint32_t totalErrCounter = 0U;
```

```

hardware_init();
dbg_uart_init();

PRINTF("\r\nCADC PD driver Test: Start...\r\n");

for (idx = 0U; idx < 1; idx++)
{
    printf("CADC instance: %d.\r\n", idx);
    errCounter = 0U;

    errCounter += CADC_DRV_TEST_PollingMode(idx);

    PRINTF("CADC %d PD driver Test error counter: %d\r\n", idx, errCounter);
    totalErrCounter += errCounter;
}

PRINTF("CADC PD driver Test: End\r\n");

// Print log for auto run detection. //
PRINTF("CADC PD driver Test ");
if (0U == totalErrCounter)
{
    PRINTF("Succeed\r\n");
}
else
{
    PRINTF("Error\r\n");
}

while ('c' != getchar()) {}
return 0;
}

static uint32_t CADC_DRV_TEST_PollingMode(uint32_t instance)
{
    volatile int tmp;
    g_CurADCInstance = instance;

    CADC_DRV_StructInitUserConfigDefault(&g_AdcUserConfigStruct);
    CADC_DRV_Init(instance, &g_AdcUserConfigStruct);

    // Configure ADC sample input channel. //
    g_AdcChnConfigStruct.diffChns = kCAdcDiffChnANA4_5;
    g_AdcChnConfigStruct.diffSelMode = kCAdcChnSelBoth;
    g_AdcChnConfigStruct.gainMode = kCAdcSGainBy1;
    CADC_DRV_ConfigSampleChn(instance, &g_AdcChnConfigStruct);

    g_AdcChnConfigStruct.diffChns = kCAdcDiffChnANA6_7;
    g_AdcChnConfigStruct.diffSelMode = kCAdcChnSelBoth;
    g_AdcChnConfigStruct.gainMode = kCAdcSGainBy1;
    CADC_DRV_ConfigSampleChn(instance, &g_AdcChnConfigStruct);

    g_AdcChnConfigStruct.diffChns = kCAdcDiffChnANB4_5;
    g_AdcChnConfigStruct.diffSelMode = kCAdcChnSelBoth;
    g_AdcChnConfigStruct.gainMode = kCAdcSGainBy1;
    CADC_DRV_ConfigSampleChn(instance, &g_AdcChnConfigStruct);

    g_AdcChnConfigStruct.diffChns = kCAdcDiffChnANB6_7;
    g_AdcChnConfigStruct.diffSelMode = kCAdcChnSelBoth;
    g_AdcChnConfigStruct.gainMode = kCAdcSGainBy1;
    CADC_DRV_ConfigSampleChn(instance, &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. //

```

CADC Peripheral Driver

```

g_AdcConvConfigStruct.clkDivValue = 0x4U;
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,
    kCAdcConvPowerDown) ) {}
while (CADC_DRV_GetConvFlag(instance, kCAdcConvB,
    kCAdcConvPowerDown) ) {}

// 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.syncIntEnable = false;
// For each slot in conversion sequence. //
// Slot 0. //
g_AdcSlotConfigStruct.diffChns = kCAdcDiffChnANA4_5;
g_AdcSlotConfigStruct.diffSel = kCAdcChnSelBoth;
g_AdcSlotConfigStruct.slotDisable = false;
CADC_DRV_ConfigSeqSlot(instance, 0U, &g_AdcSlotConfigStruct);
// Slot 1. //
g_AdcSlotConfigStruct.diffChns = kCAdcDiffChnANA6_7;
g_AdcSlotConfigStruct.diffSel = kCAdcChnSelBoth;
g_AdcSlotConfigStruct.slotDisable = false;
CADC_DRV_ConfigSeqSlot(instance, 1U, &g_AdcSlotConfigStruct);
// Slot 2. //
g_AdcSlotConfigStruct.diffChns = kCAdcDiffChnANB4_5;
g_AdcSlotConfigStruct.diffSel = kCAdcChnSelBoth;
g_AdcSlotConfigStruct.slotDisable = false;
CADC_DRV_ConfigSeqSlot(instance, 2U, &g_AdcSlotConfigStruct);
// Slot 3. //
g_AdcSlotConfigStruct.diffChns = kCAdcDiffChnANB6_7;
g_AdcSlotConfigStruct.diffSel = kCAdcChnSelBoth;
g_AdcSlotConfigStruct.slotDisable = false;
CADC_DRV_ConfigSeqSlot(instance, 3U, &g_AdcSlotConfigStruct);

g_AdcSlotConfigStruct.slotDisable = true;
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, 1U << 0U,
    kCAdcSlotReady)) { }
while (!CADC_DRV_GetSlotFlag(instance, 1U << 1U,
    kCAdcSlotReady)) { }
while (!CADC_DRV_GetSlotFlag(instance, 1U << 2U,
    kCAdcSlotReady)) { }
while (!CADC_DRV_GetSlotFlag(instance, 1U << 3U,
    kCAdcSlotReady)) { }

tmp = (int16_t)CADC_DRV_GetSeqSlotConvValue(instance, 0U);
PRINTF("ADC Slot %2d value: %d\r\n", 0U, tmp);

```

```

tmp = (int16_t)CADC_DRV_GetSeqSlotConvValue(instance, 1U);
PRINTF("ADC Slot %2d value: %d\r\n", 1U, tmp);
tmp = (int16_t)CADC_DRV_GetSeqSlotConvValue(instance, 2U);
PRINTF("ADC Slot %2d value: %d\r\n", 2U, tmp);
tmp = (int16_t)CADC_DRV_GetSeqSlotConvValue(instance, 3U);
PRINTF("ADC Slot %2d value: %d\r\n", 3U, tmp);

CADC_DRV_Deinit(instance);
return 0U;
}

```

Enumerations

- enum `cadc_conv_id_t` {
`kCAdcConvA` = 0U,
`kCAdcConvB` = 1U }
Defines the type of enumerating ADC converter ID.
- enum `cadc_flag_t` {
`kCAdcConvInProgress` = 0U,
`kCAdcConvEndOfScanInt` = 1U,
`kCAdcConvPowerDown` = 2U,
`kCAdcZeroCrossingInt` = 3U,
`kCAdcLowLimitInt` = 4U,
`kCAdcHighLimitInt` = 5U,
`kCAdcSlotReady` = 6U,
`kCAdcSlotLowLimitEvent` = 7U,
`kCAdcSlotHighLimitEvent` = 8U,
`kCAdcSlotCrossingEvent` = 9U }
Defines types for an enumerating event.

Variables

- ADC_Type * `g_cadcBaseAddr` []
Table of base addresses for ADC instances.
- IRQn_Type `g_cadcErrIrqId` [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_controller_config_t *userConfigPtr`)
Populates the user configuration structure for the CyclicADC common settings.
- `cadc_status_t CADC_DRV_Init` (`uint32_t instance`, `const cadc_controller_config_t *userConfigPtr`)
Initializes the CyclicADC module for a global configuration.
- `cadc_status_t CADC_DRV_Deinit` (`uint32_t instance`)
De-initializes the CyclicADC module.
- `cadc_status_t CADC_DRV_StructInitConvConfigDefault` (`cadc_converter_config_t *configPtr`)

CADC Peripheral Driver

- Populates the user configuration structure for each converter.*

 - `cadc_status_t CADC_DRV_ConfigConverter` (uint32_t instance, `cadc_conv_id_t` convId, const `cadc_converter_config_t` *configPtr)

Configures each converter in the CyclicADC module.

 - `cadc_status_t CADC_DRV_ConfigSampleChn` (uint32_t instance, const `cadc_chn_config_t` *configPtr)

Configures the input channel for ADC conversion.

 - `cadc_status_t CADC_DRV_ConfigSeqSlot` (uint32_t instance, uint32_t slotIdx, const `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_GetSeqSlotConvValue` (uint32_t instance, uint32_t slotIdx)

Reads the conversion value and returns an absolute 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.

 - uint16_t `CADC_DRV_GetSlotFlag` (uint32_t instance, uint16_t slotIdxMask, `cadc_flag_t` flag)

Gets the flag for each slot event.

 - void `CADC_DRV_ClearSlotFlag` (uint32_t instance, uint16_t slotIdxMask, `cadc_flag_t` flag)

Clears the flag for each slot event.

8.3.6 Enumeration Type Documentation

8.3.6.1 enum `cadc_conv_id_t`

Enumerator

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

8.3.6.2 enum `cadc_flag_t`

Enumerator

- `kCAdcConvInProgress` Conversion in progress for each converter.
- `kCAdcConvEndOfScanInt` End of scan interrupt.
- `kCAdcConvPowerDown` The converter is powered Down.
- `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.

8.3.7 Function Documentation

8.3.7.1 `cadc_status_t CADC_DRV_StructInitUserConfigDefault (cadc_controller_config_t * userConfigPtr)`

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; .dmaSrc = kCAdcDmaTriggeredByEndOfScan; .autoStandbyEnable = false; .powerUpDelayCount = 0x2AU; .autoPowerDownEnable = false;
```

Parameters

<i>userConfigPtr</i>	Pointer to structure of "cadc_controller_config_t".
----------------------	---

Returns

Execution status.

8.3.7.2 `cadc_status_t CADC_DRV_Init (uint32_t instance, const cadc_controller_config_t * userConfigPtr)`

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

Parameters

<i>instance</i>	Instance ID number.
<i>userConfigPtr</i>	Pointer to structure of "cadc_controller_config_t".

Returns

Execution status.

8.3.7.3 `cadc_status_t CADC_DRV_Deinit (uint32_t instance)`

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

CADC Peripheral Driver

Parameters

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

Returns

Execution status.

8.3.7.4 `cadc_status_t CADC_DRV_StructInitConvConfigDefault (cadc_converter_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; .clkDivValue = 0x3FU; .useChnInputAsVrefH = false; .useChnInputAsVrefL = false; .speedMode = kCAdcConvClkLimitBy25MHz; .sampleWindowCount = 0U;
```

Parameters

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

Returns

Execution status.

8.3.7.5 `cadc_status_t CADC_DRV_ConfigConverter (uint32_t instance, cadc_conv_id_t convId, const cadc_converter_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_converter_config_t".
------------------	--

Returns

Execution status.

8.3.7.6 **cadc_status_t CADC_DRV_ConfigSampleChn (uint32_t instance, const 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>configPtr</i>	Pointer to configure structure. See "cadc_chn_config_t".

Returns

Execution status.

8.3.7.7 **cadc_status_t CADC_DRV_ConfigSeqSlot (uint32_t instance, uint32_t slotIdx, const 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>slotIdx</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.

8.3.7.8 **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.

CADC Peripheral Driver

Parameters

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

8.3.7.9 uint16_t CADC_DRV_GetSeqSlotConvValue (uint32_t instance, uint32_t slotIdx)

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

Parameters

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

8.3.7.10 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.

8.3.7.11 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".
-------------	-------------------------------------

8.3.7.12 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.

8.3.7.13 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".

8.3.7.14 uint16_t CADC_DRV_GetSlotFlag (uint32_t instance, uint16_t slotIdxMask, cadc_flag_t flag)

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

Parameters

CADC Peripheral Driver

<i>instance</i>	Instance ID number.
<i>slotIdxMask</i>	Indicated slot number's mask.
<i>flag</i>	Indicated event. See "cadc_flag_t".

Returns

Assertion of indicated event.

8.3.7.15 void CADC_DRV_ClearSlotFlag (uint32_t instance, uint16_t slotIdxMask, 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>slotIdxMask</i>	Indicated slot number's mask.
<i>flag</i>	Indicated event. See "cadc_flag_t".

8.3.8 Variable Documentation

8.3.8.1 ADC_Type* g_cadcBaseAddr[]

8.3.8.2 IRQn_Type g_cadcErrIrqlId[ADC_INSTANCE_COUNT]

Chapter 9

Digital-to-Analog Converter (DAC)

9.1 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](#)

DAC HAL Driver

9.2 DAC HAL Driver

9.2.1 Overview

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

Data Structures

- struct `dac_converter_config_t`
Defines the type of structure for configuring the DAC converter. [More...](#)
- struct `dac_buffer_config_t`
Defines the type of structure for configuring the DAC buffer. [More...](#)

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` (DAC_Type *base)
Resets all configurable registers to be in the reset state for DAC.
- void `DAC_HAL_ConfigConverter` (DAC_Type *base, const `dac_converter_config_t` *configPtr)
Configures the converter for DAC.
- void `DAC_HAL_ConfigBuffer` (DAC_Type *base, const `dac_buffer_config_t` *configPtr)
Configures the buffer for DAC.

- void [DAC_HAL_SetBuffValue](#) (DAC_Type *base, uint8_t idx, uint16_t value)
Sets the 12-bit value for the DAC items in the buffer.
- static void [DAC_HAL_ClearBuffIdxUpperFlag](#) (DAC_Type *base)
Clears the flag of the DAC buffer read pointer.
- static bool [DAC_HAL_GetBuffIdxUpperFlag](#) (DAC_Type *base)
Gets the flag of the DAC buffer read pointer when it hits the bottom position.
- static void [DAC_HAL_ClearBuffIdxStartFlag](#) (DAC_Type *base)
Clears the flag of the DAC buffer read pointer when it hits the top position.
- static bool [DAC_HAL_GetBuffIdxStartFlag](#) (DAC_Type *base)
Gets the flag of the DAC buffer read pointer when it hits the top position.
- static void [DAC_HAL_Enable](#) (DAC_Type *base)
Enables the Programmable Reference Generator.
- static void [DAC_HAL_Disable](#) (DAC_Type *base)
Disables the Programmable Reference Generator.
- static void [DAC_HAL_SetSoftTriggerCmd](#) (DAC_Type *base)
Triggers the converter with software.
- static void [DAC_HAL_SetBuffCurIdx](#) (DAC_Type *base, uint8_t idx)
Sets the buffer index for the DAC module.
- static uint8_t [DAC_HAL_GetBuffCurIdx](#) (DAC_Type *base)
Gets the buffer index for the DAC module.

9.2.2 Data Structure Documentation

9.2.2.1 struct dac_converter_config_t

Data Fields

- [dac_ref_volt_src_mode_t](#) dacRefVoltSrc
Select the reference voltage source.
- bool [lowPowerEnable](#)
Enable the low power mode.

9.2.2.2 struct dac_buffer_config_t

Data Fields

- bool [bufferEnable](#)
Enable the buffer function.
- [dac_trigger_mode_t](#) triggerMode
Select the trigger mode.
- bool [idxStartIntEnable](#)
Switcher to enable interrupt when buffer index hits the start (0).
- bool [idxUpperIntEnable](#)
Switcher to enable interrupt when buffer index hits the upper limit.
- bool [dmaEnable](#)
Switcher to enable DMA request by original interrupts.
- [dac_buff_work_mode_t](#) buffWorkMode
Selection of buffer's work mode.

DAC HAL Driver

- `uint8_t upperIdx`
Setting of the buffer's upper limit, 0-15.

9.2.2.2.0.9 Field Documentation

9.2.2.2.0.9.1 `dac_buff_work_mode_t` `dac_buffer_config_t::buffWorkMode`

See "`dac_buff_work_mode_t`".

9.2.3 Enumeration Type Documentation

9.2.3.1 `enum dac_status_t`

Enumerator

- `kStatus_DAC_Success`* Success.
- `kStatus_DAC_InvalidArgument`* Invalid argument existed.
- `kStatus_DAC_Failed`* Execution failed.

9.2.3.2 `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.

9.2.3.3 `enum dac_trigger_mode_t`

Enumerator

- `kDacTriggerByHardware`* Select hardware trigger.
- `kDacTriggerBySoftware`* Select software trigger.

9.2.3.4 `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 limit of buffer.

- kDacBuffWatermarkFromUpperAs2Word* Select 2 word away from the upper limit of buffer.
- kDacBuffWatermarkFromUpperAs3Word* Select 3 word away from the upper limit of buffer.
- kDacBuffWatermarkFromUpperAs4Word* Select 4 word away from the upper limit of buffer.

9.2.3.5 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.

9.2.4 Function Documentation

9.2.4.1 void DAC_HAL_Init (DAC_Type * base)

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

9.2.4.2 void DAC_HAL_ConfigConverter (DAC_Type * base, const dac_converter_config_t * configPtr)

This function configures the converter for DAC. The features it covers are a one-time setting in the application.

DAC HAL Driver

Parameters

<i>base</i>	The DAC peripheral base address.
<i>configPtr</i>	The pointer to configure structure.

9.2.4.3 void DAC_HAL_ConfigBuffer (DAC_Type * *base*, const dac_buffer_config_t * *configPtr*)

This function configures the converter for DAC. The features it covers are used for the buffer.

Parameters

<i>base</i>	The DAC peripheral base address.
<i>configPtr</i>	The pointer to configure structure.

9.2.4.4 void DAC_HAL_SetBuffValue (DAC_Type * *base*, uint8_t *idx*, 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>base</i>	The DAC peripheral base address.
<i>idx</i>	Buffer index.
<i>value</i>	Setting value.

9.2.4.5 static void DAC_HAL_ClearBuffIdxUpperFlag (DAC_Type * *base*) [inline], [static]

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

Parameters

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

9.2.4.6 static bool DAC_HAL_GetBuffIdxUpperFlag (DAC_Type * *base*) [inline], [static]

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

Parameters

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

Returns

Assertion of indicated event.

9.2.4.7 static void DAC_HAL_ClearBuffIdxStartFlag (DAC_Type * *base*) [inline], [static]

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

Parameters

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

9.2.4.8 static bool DAC_HAL_GetBuffIdxStartFlag (DAC_Type * *base*) [inline], [static]

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

Parameters

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

Returns

Assertion of indicated event.

9.2.4.9 static void DAC_HAL_Enable (DAC_Type * *base*) [inline], [static]

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

Parameters

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

9.2.4.10 static void DAC_HAL_Disable (DAC_Type * *base*) [inline], [static]

This function disables the Programmable Reference Generator. Then, the DAC system is disabled.

DAC HAL Driver

Parameters

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

9.2.4.11 **static void DAC_HAL_SetSoftTriggerCmd (DAC_Type * *base*) [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>base</i>	The DAC peripheral base address.
-------------	----------------------------------

9.2.4.12 **static void DAC_HAL_SetBuffCurIdx (DAC_Type * *base*, uint8_t *idx*) [inline], [static]**

This function sets the current buffer index for the DAC module.

Parameters

<i>base</i>	the DAC peripheral base address.
<i>idx</i>	Setting buffer index.

9.2.4.13 **static uint8_t DAC_HAL_GetBuffCurIdx (DAC_Type * *base*) [inline], [static]**

This function gets the current buffer index for the DAC module.

Parameters

<i>base</i>	the DAC peripheral base address.
-------------	----------------------------------

Returns

Current index of buffer.

9.3 DAC Peripheral Driver

9.3.1 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.

9.3.2 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_ConfigBuffer()` function should be called to initialize the buffer.

9.3.3 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 descends by 1 in the next trigger event until it reaches and turns back.
- One-time Scan mode. In One-time Scan mode, the read pointer increases by one, after it is 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 modes. The buffer size is set to the max value by default. The previous upper index is a write pointer of the ring FIFO. The previous read pointer points to the tail of the ring FIFO. When the user pushes the data into FIFO, the write pointer increases. When the user triggers the buffer to pop the data, the read pointer increases.

DAC Peripheral Driver

9.3.4 DAC Call diagram

There are four kinds of typical use cases:

- Normal converter mode. Normal converter mode works 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 a normal ring buffer mode.
- Buffer Swing mode. Buffer Swing mode enables the internal buffer and sets it as a swing mode.
- Buffer One-time Scan mode. One-time Scan buffer mode enables the internal buffer and sets it as a one-time scan mode.
- Buffer FIFO mode. Buffer FIFO enables the internal buffer and sets it as a FIFO mode.

These use cases function either on the software trigger or the 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, uint16_t *buffPtr, uint8_t buffLen)
{
    dac_converter_config_t dacUserConfigStruct;
    dac_buffer_config_t dacBuffConfigStruct;
    uint8_t i;
    volatile uint16_t dacValue;

    // Fill the structure with configuration of software trigger. //
    DAC_DRV_StructInitUserConfigNormal(&dacUserConfigStruct);

    // Initialize the DAC Converter. //
    DAC_DRV_Init(instance, &dacUserConfigStruct);

    // Register the callback function for DAC buffer event. //
    DAC_TEST_InstallCallback(instance, DAC_ISR_Buffer);

    // Enable the feature of DAC internal buffer. //
    dacBuffConfigStruct.bufferEnable = true;
    dacBuffConfigStruct.triggerMode = kDacTriggerBySoftware;
#if FSL_FEATURE_DAC_HAS_WATERMARK_SELECTION
    dacBuffConfigStruct.idxWatermarkIntEnable = true;
    dacBuffConfigStruct.watermarkMode = kDacBuffWatermarkFromUpperAs2Word;
#endif // FSL_FEATURE_DAC_HAS_WATERMARK_SELECTION //
    dacBuffConfigStruct.idxStartIntEnable = true;
    dacBuffConfigStruct.idxUpperIntEnable = true;
    dacBuffConfigStruct.dmaEnable = false;
    dacBuffConfigStruct.buffWorkMode = kDacBuffWorkAsNormalMode;
    dacBuffConfigStruct.upperIdx = buffLen - 1U;
    DAC_DRV_ConfigBuffer(instance, &dacBuffConfigStruct);

    // Fill the buffer with setting data. //
    DAC_DRV_SetBuffValue(instance, 0, buffLen, buffPtr);

    // Trigger the buffer to output setting value. //
    PRINTF("DAC_DRV_SoftTriggerBuff:\r\n");
    for (i = 0; i < buffLen*2U; i++)
    {
        DAC_DRV_SoftTriggerBuffCmd(instance);
        PRINTF("%d: %d\r\n", i, buffPtr[DAC_DRV_GetBuffCurIdx(instance)]);
    }
}

```

DAC Peripheral Driver

```

    // 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, uint16_t *buffPtr, uint8_t buffLen)
{
    dac_converter_config_t dacUserConfigStruct;
    dac_buffer_config_t MyDacBuffConfigStruct;
    uint8_t i;
    volatile uint16_t dacValue;

    // Fill the structure with configuration of software trigger. //
    DAC_DRV_StructInitUserConfigNormal (&dacUserConfigStruct);

    // Initialize the DAC Converter. //
    DAC_DRV_Init(instance, &dacUserConfigStruct);

    // Register the callback function for DAC buffer event. //
    DAC_TEST_InstallCallback(instance, DAC_ISR_Buffer);

    // Enable the feature of DAC internal buffer. //
    MyDacBuffConfigStruct.bufferEnable = true;
    MyDacBuffConfigStruct.triggerMode = kDacTriggerBySoftware;
#if FSL_FEATURE_DAC_HAS_WATERMARK_SELECTION
    MyDacBuffConfigStruct.idxWatermarkIntEnable = true;
    MyDacBuffConfigStruct.watermarkMode =

```

```

        kDacBuffWatermarkFromUpperAs2Word;
#endif // FSL_FEATURE_DAC_HAS_WATERMARK_SELECTION //
MyDacBuffConfigStruct.idxStartIntEnable    = true;
MyDacBuffConfigStruct.idxUpperIntEnable   = true;
MyDacBuffConfigStruct.dmaEnable           = false;
MyDacBuffConfigStruct.buffWorkMode        = kDacBuffWorkAsSwingMode;
MyDacBuffConfigStruct.upperIdx            = buffLen - 1U;
DAC_DRV_ConfigBuffer(instance, &MyDacBuffConfigStruct);

// Fill the buffer with setting data. //
for (i = 0; i < buffLen; i++)
{
    DAC_DRV_SetBuffValue(instance, 0, buffLen, buffPtr);
}

// Trigger the buffer to output setting value. //
PRINTF("DAC_DRV_SoftTriggerBuff:\r\n");
for (i = 0; i < buffLen*2U; i++)
{
    DAC_DRV_SoftTriggerBuffCmd(instance);

    PRINTF("%d: %d\r\n", i, buffPtr[DAC_DRV_GetBuffCurIdx(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;
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);

```

DAC Peripheral Driver

```

void DAC_TEST_BufferOneTimeScanMode(uint32_t instance, uint16_t *buffPtr, uint8_t buffLen)
{
    dac_converter_config_t MyDacUserConfigStruct;
    dac_buffer_config_t MyDacBuffConfigStruct;
    uint8_t i;
    volatile uint16_t dacValue;

    // Fill the structure with configuration of software trigger. //
    DAC_DRV_StructInitUserConfigNormal(&MyDacUserConfigStruct);

    // Initialize the DAC Converter. //
    DAC_DRV_Init(instance, &MyDacUserConfigStruct);

    // Register the callback function for DAC buffer event. //
    DAC_TEST_InstallCallback(instance, DAC_ISR_Buffer);

    // Enable the feature of DAC internal buffer. //
    MyDacBuffConfigStruct.bufferEnable = true;
    MyDacBuffConfigStruct.triggerMode = kDacTriggerBySoftware;
#ifdef FSL_FEATURE_DAC_HAS_WATERMARK_SELECTION
    MyDacBuffConfigStruct.idxWatermarkIntEnable = true;
    MyDacBuffConfigStruct.watermarkMode = kDacBuffWatermarkFromUpperAs2Word
    ;
#endif // FSL_FEATURE_DAC_HAS_WATERMARK_DETECTION //
    MyDacBuffConfigStruct.idxStartIntEnable = true;
    MyDacBuffConfigStruct.idxUpperIntEnable = true;
    MyDacBuffConfigStruct.dmaEnable = false;
    MyDacBuffConfigStruct.buffWorkMode = kDacBuffWorkAsOneTimeScanMode;
    MyDacBuffConfigStruct.upperIdx = buffLen - 1U;
    DAC_DRV_ConfigBuffer(instance, &MyDacBuffConfigStruct);

    // Fill the buffer with setting data. //
    DAC_DRV_SetBuffValue(instance, 0, buffLen, buffPtr);

    // Trigger the buffer to output setting value. //
    PRINTF("DAC_DRV_SoftTriggerBuff:\r\n");
    for (i = 0; i < buffLen*2U; i++)
    {
        DAC_DRV_SoftTriggerBuffCmd(instance);
        PRINTF("%d: %d\r\n", i, buffPtr[DAC_DRV_GetBuffCurIdx(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, uint16_t *buffPtr, uint8_t buffLen)
{
    dac_converter_config_t dacUserConfigStruct;
    dac_buffer_config_t dacBuffConfigStruct;
    uint8_t i;
    volatile uint16_t dacValue;

    // Fill the structure with configuration of software trigger. //
    DAC_DRV_StructInitUserConfigNormal(&dacUserConfigStruct);

    // Initialize the DAC Converter. //
    DAC_DRV_Init(instance, &dacUserConfigStruct);
    // Enable the feature of DAC internal buffer. //
    dacBuffConfigStruct.bufferEnable = true;
    dacBuffConfigStruct.triggerMode = kDacTriggerBySoftware;
#if FSL_FEATURE_DAC_HAS_WATERMARK_SELECTION
    dacBuffConfigStruct.idxWatermarkIntEnable = false;
    dacBuffConfigStruct.watermarkMode =
        kDacBuffWatermarkFromUpperAs2Word;
#endif // FSL_FEATURE_DAC_HAS_WATERMARK_SELECTION //
    dacBuffConfigStruct.idxStartIntEnable = false;
    dacBuffConfigStruct.idxUpperIntEnable = false;
    dacBuffConfigStruct.dmaEnable = false;
    dacBuffConfigStruct.buffWorkMode = kDacBuffWorkAsFIFOMode;
    dacBuffConfigStruct.upperIdx = 0; // Write Pointer. //
    DAC_DRV_ConfigBuffer(instance, &dacBuffConfigStruct);

    // Fill the FIFO with setting data. //
    for (i = 0; i < buffLen; i++)
    {
        DAC_DRV_SetBuffValue(instance, 0, 1U, &buffPtr[i]);
    }

    // Trigger the buffer to output setting value. //
    PRINTF("DAC_DRV_SoftTriggerBuff:\r\n");
    for (i = 0; i < buffLen*2U; i++)
    {
        DAC_DRV_SoftTriggerBuffCmd(instance);
        PRINTF("%d: %d\r\n", i, buffPtr[DAC_DRV_GetBuffCurIdx(instance)]);
    }

    // De-initialize the DAC converter. //
    DAC_DRV_Deinit(instance);
}

static void DAC_ISR_Buffer(void)

```

DAC Peripheral Driver

```

{
    if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexStartFlag) )
    {
        MyDacIsrCounterBuffStart++;
    }
    if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexWatermarkFlag) )
    {
        // When the FIFO's data count is less than the watermark, it would be filled here automatically.
        //
        if (g_dacBuffIndex >= DAC_TEST_BUFF_SIZE)
        {
            g_dacBuffIndex = 0U;
        }
        DAC_DRV_SetBuffValue(g_dacInstance, 0U, 1U, &g_dacBuffDat[g_dacBuffIndex++]);
        MyDacIsrCounterBuffWatermark++;
    }
    if ( DAC_DRV_GetFlag(g_dacInstance, kDacBuffIndexUpperFlag) )
    {
        MyDacIsrCounterBuffUpper++;
    }
}

```

Enumerations

- enum `dac_flag_t` {
`kDacBuffIndexStartFlag = 1U`,
`kDacBuffIndexUpperFlag = 2U` }
Defines the type of event flags.

Functions

- `dac_status_t DAC_DRV_StructInitUserConfigNormal` (`dac_converter_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`, `const dac_converter_config_t *userConfigPtr`)
Initializes the converter.
- `dac_status_t 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_ConfigBuffer` (`uint32_t instance`, `const dac_buffer_config_t *configPtr`)
Configures 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.
- `void DAC_DRV_SoftTriggerBuffCmd` (`uint32_t instance`)
Triggers the buffer by software and returns the current value.
- `void DAC_DRV_ClearBuffFlag` (`uint32_t instance`, `dac_flag_t flag`)
Clears the flag for an indicated event causing an interrupt.
- `bool DAC_DRV_GetBuffFlag` (`uint32_t instance`, `dac_flag_t flag`)
Gets the flag for an indicated event causing an interrupt.
- `void DAC_DRV_SetBuffCurIdx` (`uint32_t instance`, `uint8_t idx`)
Sets the current read pointer in DAC buffer.
- `uint8_t DAC_DRV_GetBuffCurIdx` (`uint32_t instance`)

Gets the current read pointer in the DAC buffer.

Variables

- DAC_Type *const [g_dacBase](#) []
Table of base addresses for DAC instances.
- const IRQn_Type [g_dacIrqId](#) [DAC_INSTANCE_COUNT]
Table to save DAC IRQ enumeration numbers defined in the CMSIS header file.

9.3.5 Enumeration Type Documentation

9.3.5.1 enum dac_flag_t

Enumerator

kDacBuffIndexStartFlag Event for the buffer index hit the start (0).

kDacBuffIndexUpperFlag Event for the buffer index hit the upper.

9.3.6 Function Documentation

9.3.6.1 dac_status_t DAC_DRV_StructInitUserConfigNormal (dac_converter_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.

9.3.6.2 dac_status_t DAC_DRV_Init (uint32_t instance, const dac_converter_config_t * userConfigPtr)

This function initializes the converter.

DAC Peripheral Driver

Parameters

<i>instance</i>	DAC instance ID.
<i>userConfigPtr</i>	Pointer to the initialization structure. See the "dac_user_config_t".

Returns

Execution status.

9.3.6.3 `dac_status_t 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.

Parameters

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

Returns

Execution status.

9.3.6.4 `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.

9.3.6.5 `dac_status_t DAC_DRV_ConfigBuffer (uint32_t instance, const dac_buffer_config_t * configPtr)`

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>configPtr</i>	Pointer to the configuration structure. See the "dac_buff_config_t".

Returns

Execution status.

9.3.6.6 **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.

9.3.6.7 **void DAC_DRV_SoftTriggerBuffCmd (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.
-----------------	------------------

9.3.6.8 **void DAC_DRV_ClearBuffFlag (uint32_t instance, dac_flag_t flag)**

This function clears the flag for an indicated event causing an interrupt.

DAC Peripheral Driver

Parameters

<i>instance</i>	DAC instance ID.
<i>flag</i>	Indicated flag. See "dac_flag_t".

9.3.6.9 bool DAC_DRV_GetBuffFlag (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.

9.3.6.10 void DAC_DRV_SetBuffCurIdx (uint32_t instance, uint8_t idx)

This function sets the current read pointer in DAC buffer.

Parameters

<i>instance</i>	DAC instance ID.
<i>idx</i>	Index for read pointer in buffer.

9.3.6.11 uint8_t DAC_DRV_GetBuffCurIdx (uint32_t instance)

This function gets the current read pointer in DAC buffer.

Parameters

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

Returns

Index for current read pointer in buffer.

9.3.7 Variable Documentation

9.3.7.1 DAC_Type* const g_dacBase[]

9.3.7.2 const IRQn_Type g_dacIrqId[DAC_INSTANCE_COUNT**]**

Chapter 10

Direct Memory Access (DMA)

10.1 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

10.2 DMA HAL driver

10.2.1 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...](#)
- struct [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,
[kDmaChannelLinkChan1AfterBCRO](#) = 0x3U }
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](#) (DMA_Type *base, uint32_t channel)
Sets all registers of the channel to 0.
- void [DMA_HAL_ConfigTransfer](#) (DMA_Type *base, uint32_t channel, [dma_transfer_size_t](#) size, [dma_transfer_type_t](#) type, uint32_t sourceAddr, uint32_t destAddr, uint32_t length)

- Basic DMA transfer configuration.*

 - static void [DMA_HAL_SetSourceAddr](#) (DMA_Type *base, uint32_t channel, uint32_t address)
Configures the source address.
 - static void [DMA_HAL_SetDestAddr](#) (DMA_Type *base, uint32_t channel, uint32_t address)
Configures the source address.
 - static void [DMA_HAL_SetTransferCount](#) (DMA_Type *base, uint32_t channel, uint32_t count)
Configures the bytes to be transferred.
 - static uint32_t [DMA_HAL_GetUnfinishedByte](#) (DMA_Type *base, uint32_t channel)
Gets the left bytes not to be transferred.
 - static void [DMA_HAL_SetIntCmd](#) (DMA_Type *base, uint8_t channel, bool enable)
Enables the interrupt for the DMA channel after the work is done.
 - static void [DMA_HAL_SetCycleStealCmd](#) (DMA_Type *base, uint8_t channel, bool enable)
Configures the DMA transfer mode to cycle steal or continuous modes.
 - static void [DMA_HAL_SetAutoAlignCmd](#) (DMA_Type *base, uint8_t channel, bool enable)
Configures the auto-align feature.
 - static void [DMA_HAL_SetAsyncDmaRequestCmd](#) (DMA_Type *base, uint8_t channel, bool enable)
Configures the a-sync DMA request feature.
 - static void [DMA_HAL_SetSourceIncrementCmd](#) (DMA_Type *base, uint32_t channel, bool enable)
Enables/disables the source increment.
 - static void [DMA_HAL_SetDestIncrementCmd](#) (DMA_Type *base, uint32_t channel, bool enable)
Enables/disables destination increment.
 - static void [DMA_HAL_SetSourceTransferSize](#) (DMA_Type *base, uint32_t channel, [dma_transfer_size_t](#) transfersize)
Configures the source transfer size.
 - static void [DMA_HAL_SetDestTransferSize](#) (DMA_Type *base, uint32_t channel, [dma_transfer_size_t](#) transfersize)
Configures the destination transfer size.
 - static void [DMA_HAL_SetTriggerStartCmd](#) (DMA_Type *base, uint32_t channel, bool enable)
Triggers the start.
 - static void [DMA_HAL_SetSourceModulo](#) (DMA_Type *base, uint32_t channel, [dma_modulo_t](#) modulo)
Configures the modulo for the source address.
 - static void [DMA_HAL_SetDestModulo](#) (DMA_Type *base, uint32_t channel, [dma_modulo_t](#) modulo)
Configures the modulo for the destination address.
 - static void [DMA_HAL_SetDmaRequestCmd](#) (DMA_Type *base, uint32_t channel, bool enable)
Enables/disables the DMA request.
 - static void [DMA_HAL_SetDisableRequestAfterDoneCmd](#) (DMA_Type *base, uint32_t channel, bool enable)
Configures the DMA request state after the work is done.
 - void [DMA_HAL_SetChanLink](#) (DMA_Type *base, uint8_t channel, [dma_channel_link_config_t](#) *mode)
Configures the channel link feature.
 - static void [DMA_HAL_ClearStatus](#) (DMA_Type *base, uint8_t channel)
Clears the status of the DMA channel.
 - [dma_error_status_t](#) [DMA_HAL_GetStatus](#) (DMA_Type *base, uint8_t channel)
Gets the DMA controller channel status.

DMA HAL driver

10.2.2 Data Structure Documentation

10.2.2.1 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.

10.2.2.2 struct dma_error_status_t

Data Fields

- [uint32_t dmaBytesToBeTransffered](#)
Bytes to be transferred.
- [bool dmaTransDone](#)
DMA channel transfer is done.
- [bool dmaBusy](#)
DMA is running.
- [bool dmaPendingRequest](#)
A transfer remains.
- [bool dmaDestBusError](#)
Bus error on destination address.
- [bool dmaSourceBusError](#)
Bus error on source address.
- [bool dmaConfigError](#)
Configuration error.

10.2.2.2.0.10 Field Documentation

10.2.2.2.0.10.1 [bool dma_error_status_t::dmaTransDone](#)

10.2.2.2.0.10.2 [bool dma_error_status_t::dmaBusy](#)

10.2.2.2.0.10.3 [bool dma_error_status_t::dmaPendingRequest](#)

10.2.3 Enumeration Type Documentation

10.2.3.1 enum dma_status_t

Enumerator

kStatus_DMA_InvalidArgument Parameter is not available for the current configuration.

kStatus_DMA_Fail Function operation failed.

10.2.3.2 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* 16 bits are transferred for every read/write

10.2.3.3 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 channel 1 after the BCR decrements to zero.

10.2.3.4 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.

10.2.4 Function Documentation

10.2.4.1 void DMA_HAL_Init (DMA_Type * *base*, uint32_t *channel*)

Parameters

<i>base</i>	DMA base.
<i>channel</i>	DMA channel.

10.2.4.2 void DMA_HAL_ConfigTransfer (DMA_Type * *base*, uint32_t *channel*, dma_transfer_size_t *size*, dma_transfer_type_t *type*, uint32_t *sourceAddr*, uint32_t *destAddr*, uint32_t *length*)

DMA HAL driver

Parameters

<i>base</i>	DMA base.
<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.
<i>type</i>	Transfer type.
<i>sourceAddr</i>	Source address.
<i>destAddr</i>	Destination address.
<i>length</i>	Bytes to be transferred.

10.2.4.3 `static void DMA_HAL_SetSourceAddr (DMA_Type * base, uint32_t channel, uint32_t address) [inline], [static]`

Each SAR contains the byte address used by the DMA to read data. The SAR_n 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>base</i>	DMA base.
<i>channel</i>	DMA channel.
<i>address</i>	memory address pointing to the source address.

10.2.4.4 `static void DMA_HAL_SetDestAddr (DMA_Type * base, uint32_t channel, uint32_t address) [inline], [static]`

Each DAR contains the byte address used by the DMA to read data. The DAR_n 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>base</i>	DMA base.
<i>channel</i>	DMA channel.
<i>address</i>	Destination address.

10.2.4.5 static void DMA_HAL_SetTransferCount (DMA_Type * *base*, 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>base</i>	DMA base.
<i>channel</i>	DMA channel.
<i>count</i>	bytes to be transferred.

10.2.4.6 static uint32_t DMA_HAL_GetUnfinishedByte (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

<i>base</i>	DMA base.
<i>channel</i>	DMA channel.

Returns

unfinished bytes.

10.2.4.7 static void DMA_HAL_SetIntCmd (DMA_Type * *base*, uint8_t *channel*, bool *enable*) [inline], [static]

This function enables the request for specified channel.

DMA HAL driver

Parameters

<i>base</i>	DMA base.
<i>channel</i>	DMA channel.
<i>enable</i>	True means enable interrupt, false means disable.

10.2.4.8 `static void DMA_HAL_SetCycleStealCmd (DMA_Type * base, 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>base</i>	DMA base.
<i>channel</i>	DMA channel.
<i>enable</i>	1 means cycle-steal mode, 0 means continuous mode.

10.2.4.9 `static void DMA_HAL_SetAutoAlignCmd (DMA_Type * base, 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>base</i>	DMA base.
<i>channel</i>	DMA channel.
<i>enable</i>	0 means disable auto-align. 1 means enable auto-align.

10.2.4.10 `static void DMA_HAL_SetAsyncDmaRequestCmd (DMA_Type * base, uint8_t channel, bool enable) [inline], [static]`

Enables/disables the a-synchronization mode in a STOP mode for each DMA channel.

Parameters

<i>base</i>	DMA base.
<i>channel</i>	DMA channel.
<i>enable</i>	0 means disable DMA request a-sync. 1 means enable DMA request -.

10.2.4.11 static void DMA_HAL_SetSourceIncrementCmd (DMA_Type * *base*, 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>base</i>	DMA base.
<i>channel</i>	DMA channel.
<i>enable</i>	Enabled/Disable increment.

10.2.4.12 static void DMA_HAL_SetDestIncrementCmd (DMA_Type * *base*, 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>base</i>	DMA base.
<i>channel</i>	DMA channel.
<i>enable</i>	Enabled/Disable increment.

10.2.4.13 static void DMA_HAL_SetSourceTransferSize (DMA_Type * *base*, uint32_t *channel*, dma_transfer_size_t *transfersize*) [inline], [static]

Parameters

<i>base</i>	DMA base.
-------------	-----------

DMA HAL driver

<i>channel</i>	DMA channel.
<i>transfersize</i>	enum type for transfer size.

10.2.4.14 `static void DMA_HAL_SetDestTransferSize (DMA_Type * base, uint32_t channel, dma_transfer_size_t transfersize) [inline], [static]`

Parameters

<i>base</i>	DMA base.
<i>channel</i>	DMA channel.
<i>transfersize</i>	enum type for transfer size.

10.2.4.15 `static void DMA_HAL_SetTriggerStartCmd (DMA_Type * base, 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>base</i>	DMA base.
<i>channel</i>	DMA channel.
<i>enable</i>	Enable/disable trigger start.

10.2.4.16 `static void DMA_HAL_SetSourceModulo (DMA_Type * base, uint32_t channel, dma_modulo_t modulo) [inline], [static]`

Parameters

<i>base</i>	DMA base.
<i>channel</i>	DMA channel.
<i>modulo</i>	enum data type for source modulo.

10.2.4.17 `static void DMA_HAL_SetDestModulo (DMA_Type * base, uint32_t channel, dma_modulo_t modulo) [inline], [static]`

Parameters

<i>base</i>	DMA base.
<i>channel</i>	DMA channel.
<i>modulo</i>	enum data type for dest modulo.

10.2.4.18 `static void DMA_HAL_SetDmaRequestCmd (DMA_Type * base, uint32_t channel, bool enable) [inline], [static]`

Parameters

<i>base</i>	DMA base.
<i>channel</i>	DMA channel.
<i>enable</i>	Enable/disable dma request.

10.2.4.19 `static void DMA_HAL_SetDisableRequestAfterDoneCmd (DMA_Type * base, 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>base</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.

10.2.4.20 `void DMA_HAL_SetChanLink (DMA_Type * base, uint8_t channel, dma_channel_link_config_t * mode)`

Parameters

<i>base</i>	DMA base address.
-------------	-------------------

DMA HAL driver

<i>channel</i>	DMA channel.
<i>mode</i>	Mode of channel link in DMA.

10.2.4.21 static void DMA_HAL_ClearStatus (DMA_Type * *base*, 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>base</i>	DMA base address.
<i>channel</i>	DMA channel.

10.2.4.22 dma_error_status_t DMA_HAL_GetStatus (DMA_Type * *base*, uint8_t *channel*)

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>base</i>	DMA base address.
<i>channel</i>	DMA channel.

Returns

Status of the DMA channel.

10.3 DMAMUX HAL driver

This chapter describes the programming interface of the DMAMUX HAL module.

DMA driver

10.4 DMA driver

10.4.1 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

- DMA_Type *const [g_dmaBase](#) [DMA_INSTANCE_COUNT]
Array for eDMA module register base address.
- DMAMUX_Type *const [g_dmamuxBase](#) [DMAMUX_INSTANCE_COUNT]
Array for DMAMUX module register base address.
- const IRQn_Type [g_dmaIrqId](#) [DMA_INSTANCE_COUNT][FSL_FEATURE_DMA_DMAMUX_CHANNELS]
Two-dimensional array for EDMA channel interrupt vector number.

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.

10.4.2 Data Structure Documentation

10.4.2.1 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.

DMA driver

- volatile `dma_channel_status_t` `status`
Channel status.

10.4.2.2 struct `dma_state_t`

10.4.3 Typedef Documentation

10.4.3.1 typedef `void(* dma_callback_t)(void *parameter, dma_channel_status_t status)`

A prototype for the callback function registered into the DMA driver.

10.4.4 Enumeration Type Documentation

10.4.4.1 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.

10.4.4.2 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.

10.4.5 Function Documentation

10.4.5.1 `dma_status_t` `DMA_DRV_Init (dma_state_t * state)`

Parameters

<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.

10.4.5.2 `dma_status_t DMA_DRV_Deinit (void)`

Returns

If successful, returns the `kStatus_DMA_Success`. Otherwise, it returns an error.

10.4.5.3 `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.

10.4.5.4 `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.

10.4.5.5 `dma_status_t DMA_DRV_ClaimChannel (uint32_t channel, dma_request_source_t source, dma_channel_t * chn)`

DMA driver

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.

10.4.5.6 `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 <code>kDmaAnyChannel</code> , 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.

10.4.5.7 `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.

Parameters

<i>chn</i>	Memory pointing to DMA channel.
------------	---------------------------------

Returns

If successful, returns the `kStatus_DMA_Success`. Otherwise, it returns an error.

10.4.5.8 `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.

10.4.5.9 `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.

**10.4.5.10 `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.

DMA driver

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.

10.4.5.11 `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.

10.4.5.12 `void DMA_DRV_IRQhandler (uint32_t channel)`

Parameters

<i>channel</i>	DMA channel number.
----------------	---------------------

10.4.6 Variable Documentation

10.4.6.1 `DMA_Type* const g_dmaBase[DMA_INSTANCE_COUNT]`

10.4.6.2 `DMAMUX_Type* const g_dmamuxBase[DMAMUX_INSTANCE_COUNT]`

10.4.6.3 `const IRQn_Type g_dmalrqld[DMA_INSTANCE_COUNT][FSL_FEATURE_DMA_DMAMUX_CHANNELS]`

10.5 DMA request

This section provides the DMA request resource.

10.5.1 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.

10.5.2 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.

10.5.3 DMA request concept

DMA request triggers a DMA transfer. The DMA request table is available in the device configuration chapters in a Reference Manual.

10.5.4 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.

10.5.5 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:

DMA request

```

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("Fali 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);

```

Chapter 11

Serial Peripheral Interface (DSPI)

11.1 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](#)

DSPI HAL driver

11.2 DSPI HAL driver

11.2.1 Overview

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

Files

- file [fsl_dspi_hal.h](#)

Data Structures

- struct [dspi_data_format_config_t](#)
DSPI data format settings configuration structure. [More...](#)
- struct [dspi_baud_rate_divisors_t](#)
DSPI baud rate divisors settings configuration structure. [More...](#)
- struct [dspi_command_config_t](#)
DSPI command and data configuration structure. [More...](#)

Enumerations

- enum [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,
kStatus_DSPI_EdmaStcdUnaligned32Error }
Error codes for the DSPI driver.
- enum [dspi_master_slave_mode_t](#) {
kDspiMaster = 1,
kDspiSlave = 0 }
DSPI master or slave configuration.
- enum [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_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` = SPI_RSER_TCF_RE_SHIFT,
`kDspiTxAndRxStatus` = SPI_SR_TXRXS_SHIFT,
`kDspiEndOfQueue` = SPI_RSER_EOQF_RE_SHIFT,
`kDspiTxFifoUnderflow` = SPI_RSER_TFUF_RE_SHIFT,
`kDspiTxFifoFillRequest` = SPI_RSER_TFFF_RE_SHIFT,
`kDspiRxFifoOverflow` = SPI_RSER_RFOF_RE_SHIFT,
`kDspiRxFifoDrainRequest` = SPI_RSER_RFDF_RE_SHIFT }
- DSPI status flags and interrupt request enable.*

 - enum `dspi_delay_type_t` {
`kDspiPcsToSck` = 1,
`kDspiLastSckToPcs` = 2,
`kDspiAfterTransfer` = 3 }

DSPI HAL driver

DSPI delay type selection.

Configuration

- void **DSPI_HAL_Init** (SPI_Type *base)
Restores the DSPI to reset the configuration.
- static void **DSPI_HAL_Enable** (SPI_Type *base)
Enables the DSPI peripheral and sets the MCR MDIS to 0.
- static void **DSPI_HAL_Disable** (SPI_Type *base)
Disables the DSPI peripheral, sets MCR MDIS to 1.
- uint32_t **DSPI_HAL_SetBaudRate** (SPI_Type *base, **dspi_ctar_selection_t** whichCtar, uint32_t bitsPerSec, uint32_t sourceClockInHz)
Sets the DSPI baud rate in bits per second.
- void **DSPI_HAL_SetBaudDivisors** (SPI_Type *base, **dspi_ctar_selection_t** whichCtar, const **dspi_baud_rate_divisors_t** *divisors)
Configures the baud rate divisors manually.
- static void **DSPI_HAL_SetMasterSlaveMode** (SPI_Type *base, **dspi_master_slave_mode_t** mode)
Configures the DSPI for master or slave.
- static bool **DSPI_HAL_IsMaster** (SPI_Type *base)
Returns whether the DSPI module is in master mode.
- static void **DSPI_HAL_SetContinuousSckCmd** (SPI_Type *base, bool enable)
Configures the DSPI for the continuous SCK operation.
- static void **DSPI_HAL_SetRxFifoOverwriteCmd** (SPI_Type *base, bool enable)
Configures the DSPI received FIFO overflow overwrite enable.
- void **DSPI_HAL_SetPcsPolarityMode** (SPI_Type *base, **dspi_which_pcs_config_t** pcs, **dspi_pcs_polarity_config_t** activeLowOrHigh)
Configures the DSPI peripheral chip select polarity.
- void **DSPI_HAL_SetFifoCmd** (SPI_Type *base, bool enableTxFifo, bool enableRxFifo)
Enables (or disables) the DSPI FIFOs.
- void **DSPI_HAL_SetFlushFifoCmd** (SPI_Type *base, bool enableFlushTxFifo, bool enableFlushRxFifo)
Flushes the DSPI FIFOs.
- static void **DSPI_HAL_SetDatainSamplepointMode** (SPI_Type *base, **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** (SPI_Type *base)
Starts the DSPI transfers, clears HALT bit in MCR.
- static void **DSPI_HAL_StopTransfer** (SPI_Type *base)
Stops (halts) DSPI transfers, sets HALT bit in MCR.
- **dspi_status_t** **DSPI_HAL_SetDataFormat** (SPI_Type *base, **dspi_ctar_selection_t** whichCtar, const **dspi_data_format_config_t** *config)
Configures the data format for a particular CTAR.
- void **DSPI_HAL_SetDelay** (SPI_Type *base, **dspi_ctar_selection_t** whichCtar, uint32_t prescaler, uint32_t scaler, **dspi_delay_type_t** whichDelay)
Manually configures the delay prescaler and scaler for a particular CTAR.
- uint32_t **DSPI_HAL_CalculateDelay** (SPI_Type *base, **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** (SPI_Type *base)

- Gets the DSPI master PUSHHR data register address for DMA operation.*
 - static uint32_t **DSPI_HAL_GetSlavePushrRegAddr** (SPI_Type *base)
- Gets the DSPI slave PUSHHR data register address for DMA operation.*
 - static uint32_t **DSPI_HAL_GetPoprRegAddr** (SPI_Type *base)
- Gets the DSPI POPR data register address for DMA operation.*

Interrupts

- void **DSPI_HAL_SetTxFifoFillDmaIntMode** (SPI_Type *base, **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** (SPI_Type *base, **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** (SPI_Type *base, **dspi_status_and_interrupt_request_t** interruptSrc, bool enable)
 - Configures the DSPI interrupts.*
- static bool **DSPI_HAL_GetIntMode** (SPI_Type *base, **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** (SPI_Type *base, **dspi_status_and_interrupt_request_t** statusFlag)
 - Gets the DSPI status flag state.*
- static void **DSPI_HAL_ClearStatusFlag** (SPI_Type *base, **dspi_status_and_interrupt_request_t** statusFlag)
 - Clears the DSPI status flag.*

Data transfer

- static uint32_t **DSPI_HAL_ReadData** (SPI_Type *base)
 - Reads data from the data buffer.*
- static void **DSPI_HAL_WriteDataSlavemode** (SPI_Type *base, uint32_t data)
 - Writes data into the data buffer, slave mode.*
- void **DSPI_HAL_WriteDataSlavemodeBlocking** (SPI_Type *base, uint32_t data)
 - Writes data into the data buffer, slave mode and waits till data was transmitted and return.*
- void **DSPI_HAL_WriteDataMastermode** (SPI_Type *base, **dspi_command_config_t** *command, uint16_t data)
 - Writes data into the data buffer, master mode.*
- void **DSPI_HAL_WriteDataMastermodeBlocking** (SPI_Type *base, **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** (SPI_Type *base, uint32_t data)
 - Writes a 32-bit data word (16-bit command appended with 16-bit data) into the data buffer, master mode.*

DSPI HAL driver

- void [DSPI_HAL_WriteCmdDataMastermodeBlocking](#) (SPI_Type *base, uint32_t data)
Writes a 32-bit data word (16-bit command appended with 16-bit data) into the data buffer, master mode and waits till complete to return.
- static uint32_t [DSPI_HAL_GetTransferCount](#) (SPI_Type *base)
Gets the transfer count.
- static void [DSPI_HAL_PresetTransferCount](#) (SPI_Type *base, uint16_t presetValue)
Pre-sets the transfer count.
- uint32_t [DSPI_HAL_GetFormattedCommand](#) (SPI_Type *base, [dspi_command_config_t](#) *command)
Returns the DSPI command word formatted to the PUSHHR data register bit field.

11.2.2 Data Structure Documentation

11.2.2.1 struct 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.
- [dspi_clock_polarity_t](#) [clkPolarity](#)
Active high or low clock polarity.
- [dspi_clock_phase_t](#) [clkPhase](#)
Clock phase setting to change and capture data.
- [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.

11.2.2.2 struct 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.

11.2.2.3 struct `dspi_command_config_t`

Note: This structure is used with the PUSH 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 PUSH register 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 only 16-bit data may be written (this may be contrary to some older documentation which erroneously states that a 32-bit value may be written).

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.

11.2.3 Enumeration Type Documentation

11.2.3.1 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.
- `kStatus_DSPI_InvalidParameter`** DSPI invalid parameter error.
- `kStatus_DSPI_NonInit`** DSPI driver does not initialize, not ready.
- `kStatus_DSPI_Initialized`** DSPI driver has initialized, cannot re-initialize.
- `kStatus_DSPI_DMACHannelInvalid`** DSPI driver could not request DMA channel(s)
- `kStatus_DSPI_Error`** DSPI driver error.
- `kStatus_DSPI_EdmaStcdUnaligned32Error`** DSPI Edma driver STCD unaligned to 32byte error.

DSPI HAL driver

11.2.3.2 enum dspi_master_slave_mode_t

Enumerator

kDspiMaster DSPI peripheral operates in master mode.

kDspiSlave DSPI peripheral operates in slave mode.

11.2.3.3 enum dspi_clock_polarity_t

Enumerator

kDspiClockPolarity_ActiveHigh Active-high DSPI clock (idles low)

kDspiClockPolarity_ActiveLow Active-low DSPI clock (idles high)

11.2.3.4 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.

11.2.3.5 enum dspi_shift_direction_t

Enumerator

kDspiMsbFirst Data transfers start with most significant bit.

kDspiLsbFirst Data transfers start with least significant bit.

11.2.3.6 enum dspi_ctar_selection_t

Enumerator

kDspiCtar0 CTAR0 selection option for master or slave mode.

kDspiCtar1 CTAR1 selection option for master mode only.

11.2.3.7 enum dspi_pcs_polarity_config_t

Enumerator

kDspiPcs_ActiveHigh PCS Active High (idles low)

kDspiPcs_ActiveLow PCS Active Low (idles high)

11.2.3.8 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].

11.2.3.9 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

11.2.3.10 enum dspi_dma_or_int_mode_t

Enumerator

kDspiGenerateIntReq Desired flag generates an Interrupt request.
kDspiGenerateDmaReq Desired flag generates a DMA request.

11.2.3.11 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.

DSPI HAL driver

11.2.3.12 enum dspi_delay_type_t

Enumerator

- kDspiPcsToSck* PCS-to-SCK delay.
- kDspiLastSckToPcs* Last SCK edge to PCS delay.
- kDspiAfterTransfer* Delay between transfers.

11.2.4 Function Documentation

11.2.4.1 void DSPI_HAL_Init (SPI_Type * *base*)

This function basically resets all of the DSPI registers to their default setting including disabling the module.

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
-------------	---------------------------------------

11.2.4.2 static void DSPI_HAL_Enable (SPI_Type * *base*) [inline], [static]

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
-------------	---------------------------------------

11.2.4.3 static void DSPI_HAL_Disable (SPI_Type * *base*) [inline], [static]

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
-------------	---------------------------------------

11.2.4.4 uint32_t DSPI_HAL_SetBaudRate (SPI_Type * *base*, 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>base</i>	Module base pointer of type SPI_Type.
<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>sourceClockIn-Hz</i>	Module source input clock in Hertz

Returns

The actual calculated baud rate

11.2.4.5 void DSPI_HAL_SetBaudDivisors (SPI_Type * *base*, 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>base</i>	Module base pointer of type SPI_Type.
<i>whichCtar</i>	The desired Clock and Transfer Attributes Register (CTAR) of type dspi_ctar_selection_t
<i>divisors</i>	Pointer to a structure containing the user defined baud rate divisor settings

11.2.4.6 static void DSPI_HAL_SetMasterSlaveMode (SPI_Type * *base*, dspi_master_slave_mode_t *mode*) [inline], [static]

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
<i>mode</i>	Mode setting (master or slave) of type dspi_master_slave_mode_t

11.2.4.7 static bool DSPI_HAL_IsMaster (SPI_Type * *base*) [inline], [static]

DSPI HAL driver

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
-------------	---------------------------------------

Returns

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

**11.2.4.8 static void DSPI_HAL_SetContinuousSckCmd (SPI_Type * *base*, bool *enable*)
[inline], [static]**

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
<i>enable</i>	Enables (true) or disables(false) continuous SCK operation.

**11.2.4.9 static void DSPI_HAL_SetRxFifoOverwriteCmd (SPI_Type * *base*, 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>base</i>	Module base pointer of type SPI_Type.
<i>enable</i>	If enabled (true), allows incoming data to overwrite Rx FIFO contents when full, else incoming data is ignored.

11.2.4.10 void DSPI_HAL_SetPcsPolarityMode (SPI_Type * *base*, dsp_i_which_pcs_config_t *pcs*, dsp_i_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>base</i>	Module base pointer of type SPI_Type.
<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.

11.2.4.11 void DSPI_HAL_SetFifoCmd (SPI_Type * *base*, 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>base</i>	Module base pointer of type SPI_Type.
<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

11.2.4.12 void DSPI_HAL_SetFlushFifoCmd (SPI_Type * *base*, bool *enableFlushTxFifo*, bool *enableFlushRxFifo*)

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
<i>enableFlushTxFifo</i>	Flushes (true) the Tx FIFO, else do not flush (false) the Tx FIFO
<i>enableFlushRxFifo</i>	Flushes (true) the Rx FIFO, else do not flush (false) the Rx FIFO

11.2.4.13 static void DSPI_HAL_SetDataInSamplepointMode (SPI_Type * *base*, 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.

DSPI HAL driver

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
<i>samplePnt</i>	selects when the data in (SIN) is sampled, of type dsp_i_master_sample_point_t. This value selects either 0, 1, or 2 system clocks between the SCK edge and the SIN (data in) sample.

11.2.4.14 static void DSPI_HAL_StartTransfer (SPI_Type * *base*) [inline], [static]

This function call called whenever the module is ready to begin data transfers in either master or slave mode.

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
-------------	---------------------------------------

11.2.4.15 static void DSPI_HAL_StopTransfer (SPI_Type * *base*) [inline], [static]

This function call stops data transfers in either master or slave mode.

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
-------------	---------------------------------------

11.2.4.16 dsp_i_status_t DSPI_HAL_SetDataFormat (SPI_Type * *base*, dsp_i_ctar_selection_t *whichCtar*, const dsp_i_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:

```

dsp_i_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>base</i>	Module base pointer of type SPI_Type.
<i>whichCtar</i>	The desired Clock and Transfer Attributes Register (CTAR) of type dspicatar_selection_t.
<i>config</i>	Pointer to structure containing user defined data format configuration settings.

Returns

An error code or kStatus_DSPI_Success

11.2.4.17 void DSPI_HAL_SetDelay (SPI_Type * *base*, dspicatar_selection_t *whichCtar*, uint32_t *prescaler*, uint32_t *scaler*, dspidelay_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 dspidelay_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>base</i>	Module base pointer of type SPI_Type.
<i>whichCtar</i>	The desired Clock and Transfer Attributes Register (CTAR) of type dspicatar_selection_t.
<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 dspidelay_type_t

11.2.4.18 uint32_t DSPI_HAL_CalculateDelay (SPI_Type * *base*, dspicatar_selection_t *whichCtar*, dspidelay_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 dspidelay_type_t.

The user passes which delay they want to configure along with the desired delay value in nanoseconds. The function calculates the values needed for the prescaler and scaler and returning the actual calculated

DSPI HAL driver

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. It is to the higher level peripheral driver to alert the user of an out of range delay input.

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
<i>whichCtar</i>	The desired Clock and Transfer Attributes Register (CTAR) of type dspic_tar_selection_t.
<i>whichDelay</i>	The desired delay to configure, must be of type dspic_delay_type_t
<i>sourceClockIn-Hz</i>	Module source input clock in Hertz
<i>delayInNano-Sec</i>	The desired delay value in nanoseconds.

Returns

The actual calculated delay value.

11.2.4.19 static uint32_t DSPI_HAL_GetMasterPushrRegAddr (SPI_Type * *base*) [inline], [static]

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

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
-------------	---------------------------------------

Returns

The DSPI master PUSHR data register address.

11.2.4.20 static uint32_t DSPI_HAL_GetSlavePushrRegAddr (SPI_Type * *base*) [inline], [static]

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

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
-------------	---------------------------------------

Returns

The DSPI slave PUSHR data register address.

11.2.4.21 static uint32_t DSPI_HAL_GetPoprRegAddr (SPI_Type * *base*) [inline], [static]

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

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
-------------	---------------------------------------

Returns

The DSPI POPR data register address.

11.2.4.22 void DSPI_HAL_SetTxFifoFillDmaIntMode (SPI_Type * *base*, dsp_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 dsp_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(base,
    kDspiGenerateDmaReq, true); <- to enable DMA
DSPI_HAL_SetTxFifoFillDmaIntMode(base,
    kDspiGenerateIntReq, true); <- to enable Interrupt
DSPI_HAL_SetTxFifoFillDmaIntMode(base,
    kDspiGenerateIntReq, false); <- to disable
    
```

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
-------------	---------------------------------------

DSPI HAL driver

<i>mode</i>	Configures the DSPI Tx FIFO Fill to generate an interrupt or DMA request
<i>enable</i>	Enable (true) or disable (false) the DSPI Tx FIFO Fill flag to generate requests

11.2.4.23 void DSPI_HAL_SetRxFifoDrainDmaIntMode (SPI_Type * *base*, dsp_i_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 `ds_pi_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_SetRxFifoDrainDmaIntMode(base,
    kDspiGenerateDmaReq, true); <- to enable DMA
DSPI_HAL_SetRxFifoDrainDmaIntMode(base,
    kDspiGenerateIntReq, true); <- to enable Interrupt
DSPI_HAL_SetRxFifoDrainDmaIntMode(base,
    kDspiGenerateIntReq, false); <- to disable
```

Parameters

<i>base</i>	Module base pointer of type <code>SPI_Type</code> .
<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

11.2.4.24 void DSPI_HAL_SetIntMode (SPI_Type * *base*, dsp_i_status_and_interrupt_request_t *interruptSrc*, bool *enable*)

This function configures the various interrupt sources of the DSPI. The parameters are base, 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 `ds_pi_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_SetRxFifoDrainDmaIntMode` respectively as these requests can generate either an interrupt or DMA request.

```
DSPI_HAL_SetIntMode(base, kDspiTxComplete, true); <- example use-case
```

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
<i>interruptSrc</i>	The interrupt source, of type dsp_i_status_and_interrupt_request_t
<i>enable</i>	Enable (true) or disable (false) the interrupt source to generate requests

11.2.4.25 static bool DSPI_HAL_GetIntMode (SPI_Type * *base*, dsp_i_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 base and interrupt source. It utilizes the same enumeration definitions for the interrupt sources as described in the interrupt config 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(base,
    kDspiTxComplete);
```

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
<i>interruptSrc</i>	The interrupt source, of type dsp_i_status_and_interrupt_request_t

Returns

Configuration of interrupt request: enable (true) or disable (false).

11.2.4.26 static bool DSPI_HAL_GetStatusFlag (SPI_Type * *base*, dsp_i_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 dsp_i_get_interrupt_config function.

```
getStatus = DSPI_HAL_GetStatusFlag(base, kDspiTxComplete);
```

DSPI HAL driver

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
<i>statusFlag</i>	The status flag, of type dspi_status_and_interrupt_request_t

Returns

State of the status flag: asserted (true) or not-asserted (false)

11.2.4.27 static void DSPI_HAL_ClearStatusFlag (SPI_Type * *base*, 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 base and the desired status bit to clear. The list of status bits is defined in the dspi_status_and_interrupt_request_t. The function uses these bit positions in its algorithm to clear the desired flag state. Example usage:

```
DSPI_HAL_ClearStatusFlag(base, kDspiTxComplete);
```

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
<i>statusFlag</i>	The status flag, of type dspi_status_and_interrupt_request_t

11.2.4.28 static uint32_t DSPI_HAL_ReadData (SPI_Type * *base*) [inline], [static]

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
-------------	---------------------------------------

Returns

The data from the read data buffer

11.2.4.29 static void DSPI_HAL_WriteDataSlavemode (SPI_Type * *base*, uint32_t *data*) [inline], [static]

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

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
<i>data</i>	The data to send

11.2.4.30 void DSPI_HAL_WriteDataSlavemodeBlocking (SPI_Type * *base*, uint32_t *data*)

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

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
<i>data</i>	The data to send

11.2.4.31 void DSPI_HAL_WriteDataMastermode (SPI_Type * *base*, dspicommandconfig_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:

```

dspicommandconfig_t commandConfig;
commandConfig.isChipSelectContinuous = true;
commandConfig.whichCtar = kDspiCtar0;
commandConfig.whichPcs = kDspiPcs1;
commandConfig.clearTransferCount = false;
commandConfig.isEndOfQueue = false;
DSPI_HAL_WriteDataMastermode(base, &commandConfig, dataWord);

```

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
<i>command</i>	Pointer to command structure

DSPI HAL driver

<i>data</i>	The data word to be sent
-------------	--------------------------

11.2.4.32 void DSPI_HAL_WriteDataMastermodeBlocking (SPI_Type * *base*, 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(base, &commandConfig, dataWord);
```

Note that this function does not return until after the transmit is complete. Also note that the DSPI must be enabled and running 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>base</i>	Module base pointer of type SPI_Type.
<i>command</i>	Pointer to command structure
<i>data</i>	The data word to be sent

11.2.4.33 static void DSPI_HAL_WriteCmdDataMastermode (SPI_Type * *base*, 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(base, dataWord);
```


Parameters

<i>base</i>	Module base pointer of type SPI_Type.
<i>data</i>	The data word (command and data combined) to be sent

11.2.4.34 void DSPI_HAL_WriteCmdDataMastermodeBlocking (SPI_Type * *base*, 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(base, dataWord);
```

Note that this function does not return until after the transmit is complete. Also note that the DSPI must be enabled and running 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>base</i>	Module base pointer of type SPI_Type.
<i>data</i>	The data word (command and data combined) to be sent

11.2.4.35 static uint32_t DSPI_HAL_GetTransferCount (SPI_Type * *base*) [inline], [static]

This function returns the current value of the DSPI Transfer Count Register.

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
-------------	---------------------------------------

Returns

The current transfer count

DSPI HAL driver

11.2.4.36 `static void DSPI_HAL_PresetTransferCount (SPI_Type * base, uint16_t presetValue) [inline], [static]`

This function allows the caller to pre-set the DSI Transfer Count Register to a desired value up to 65535; Incrementing past this resets the counter back to 0.

Parameters

<i>base</i>	Module base pointer of type SPI_Type.
<i>presetValue</i>	The desired pre-set value for the transfer counter

**11.2.4.37 uint32_t DSPI_HAL_GetFormattedCommand (SPI_Type * *base*,
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 PUSH 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 PUSH register. 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>base</i>	Module base pointer of type SPI_Type.
<i>command</i>	Pointer to command structure

Returns

The command word formatted to the PUSH register data register bit field

DSPI Master Driver

11.3 DSPI Master Driver

11.3.1 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.

11.3.2 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. In the subsequent chapters the enhanced DMA-driven driver is referred to as the DMA-driven driver. The interrupt-driven and DMA-driven driver APIs are distinguished with 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, these chapters refer to either set of drivers as the "DSPI master driver" when discussing items that pertain to either driver. Note, when using the DMA driven DSPI driver, initialize the eDMA module. An example is shown in the Initialization chapter.

This is a step-by-step process to initialize and transfer the SPI data. For API specific examples, see the examples below. The example 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 the SPI interrupt handler needs to be changed. The interrupt driver calls the [DSPI_DRV_IRQHandler\(\)](#) function and the eDMA driver calls the [DSPI_DRV_EdmaIRQHandler\(\)](#) function. 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 recommended to mix interrupt and DMA driven drivers in the same application. However, should the user decide to do so, separately set up and initialize another instance for DMA operations.

The user can also de-initialize the current interrupt-driven DSPI instance and re-initialize it for DMA operations. Note, that, because the DMA driven driver also uses interrupts, the user must direct the IRQ handler from the vector table to the desired driver's IRQ handler. See 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 application project to direct the DSPI interrupt vectors to the proper IRQ handlers. The `fsl_dspi_shared_function.c` and `fsl_dspi_edma_shared_function.c` files 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(base-Addr)`. Note that the interrupt driver calls the `DSPI_DRV_IRQHandler()` function and the eDMA driver calls the `DSPI_DRV_EdmaIRQHandler()` function. See files `fsl_dspi_irq.c` and `fsl_dspi_edma_irq.c` for an example of these function calls.

When using the DSPI driver with the eDMA some DSPI instances do not support separate DMA requests for TX and RX channels. In those cases with a single shared DMA request for TX and RX DMA channels, the driver links one channel to the other to still take advantage of DMA transfers. However, the drawback to using an instance with shared DMA requests limits the maximum amount of data the driver can transfer. This limit depends on the bits/frame setting.

For DSPI instances with separate TX and RX DMA requests, the maximum number of bytes that can be transferred are: 8-bit setting: 32767 bytes 16-bit setting: 65534 bytes

For DSPI instances with a shared TX and RX DMA request, the maximum number of bytes that can be transferred are: 8-bit setting: 511 bytes 16-bit setting: 1022 bytes

See the microcontroller-specific documentation to see which DSPI instance have separate or shared TX and RX DMA requests. Additionally, see the microcontroller-specific feature header file to see which DSPI instance have separate or shared TX and RX DMA requests.

11.3.3 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 the `dspi_master_state_t`. The structure for the DMA driven driver is called the `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.

11.3.4 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 the `dspi_master_user_config_t`. The user configuration structure for the DMA driven driver is called the `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 initialization function

DSPI Master Driver

(DSPI_DRV_MasterInit or DSPI_DRV_EdmaMasterInit). The user configuration structure consists of 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 settings. An example of this is provided in the Initialization section.

11.3.5 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 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 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).

11.3.6 DSPI Initialization

To initialize the DSPI master driver, call the [DSPI_DRV_MasterInit\(\)](#) or the [DSPI_DRV_EdmaMasterInit](#) functions 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 ([dspi_master_user_config_t](#) or [dspi_edma_master_user_config_t](#)). For the DMA driven driver, the memory allocation for the stdcSrc2CmdDataLast structure needs to be passed. Note that the pointer to this structure needs to be aligned to a 32-byte boundary.

First call the DSPI master initialization to initialize the DSPI module. Then, call 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. Both examples are provided below. The interrupt driven example is provided first followed by the DMA example.

This is an 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
dspi_master_state_t dspiMasterState; // simply allocate memory for this
uint32_t calculatedBaudRate;

// configure the members of the user config //
dspi_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_MasterInit(masterInstance, &dspiMasterState, &userConfig);

// Define bus configuration.
dspi_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_MasterConfigureBus(masterInstance, &spiDevice, &calculatedBaudRate);

```

This is an 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;

```

DSPI Master Driver

```
// 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 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 is initialized for master mode. The bus timing delays that can be adjusted are listed here:

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.

This function takes in as a parameter the desired delay type and the delay value (in nanoseconds) and 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);
```

11.3.7 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. The user is required to get the transfer status during the transfer to ascertain when the transfer is complete. 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 dsp_i_device_t * restrict device,
                                const uint8_t * sendBuffer,
                                uint8_t * receiveBuffer,
                                size_t transferByteCount,
                                uint32_t timeout);

dsp_i_status_t DSPI_DRV_EdmaMasterTransferBlocking(uint32_t
instance,
                                const dsp_i_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
dsp_i_status_t DSPI_DRV_MasterTransfer(uint32_t instance,
                                const dsp_i_device_t * restrict device,
                                const uint8_t * sendBuffer,
                                uint8_t * receiveBuffer,
                                size_t transferByteCount);

// Returns whether the previous transfer is completed (interrupt-driven )
dsp_i_status_t DSPI_DRV_MasterGetTransferStatus(uint32_t
instance, uint32_t * framesTransferred);

// Terminates an asynchronous transfer early (interrupt-driven )
dsp_i_status_t DSPI_DRV_MasterAbortTransfer(uint32_t instance);

// DMA-driven transfer function
dsp_i_status_t DSPI_DRV_EdmaMasterTransfer(uint32_t instance,
                                const dsp_i_edma_device_t * restrict device,
                                const uint8_t * sendBuffer,
                                uint8_t * receiveBuffer,
                                size_t transferByteCount);

// Returns whether the previous transfer is completed (DMA-driven)
dsp_i_status_t DSPI_DRV_EdmaMasterGetTransferStatus(
uint32_t instance, uint32_t * framesTransferred);

// Terminates an asynchronous transfer early (DMA-driven)
dsp_i_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);

```

DSPI Master Driver

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);

// 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);
```

11.3.8 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 [dspi_edma_device_t](#)
Data structure containing information about a device on the SPI bus with EDMA. [More...](#)
- struct [dspi_edma_master_state_t](#)
Runtime state structure for the DSPI master driver with EDMA. [More...](#)
- struct [dspi_edma_master_user_config_t](#)
The user configuration structure for the DSPI master driver with EDMA. [More...](#)
- struct [dspi_device_t](#)
Data structure containing information about a device on the SPI bus. [More...](#)
- struct [dspi_master_state_t](#)

Runtime state structure for the DSPI master driver. [More...](#)

- struct `dspi_master_user_config_t`
The user configuration structure for the DSPI master driver. [More...](#)

Variables

- SPI_Type *const `g_dspiBase` [SPI_INSTANCE_COUNT]
Table of base pointers for SPI instances.
- const IRQn_Type `g_dspiIrqId` [SPI_INSTANCE_COUNT]
Table to save DSPI IRQ enumeration numbers defined in the CMSIS header file.
- SPI_Type *const `g_dspiBase` [SPI_INSTANCE_COUNT]
Table of base pointers for SPI instances.
- const IRQn_Type `g_dspiIrqId` [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` *stcdSrc2-CmdDataLast)
Initializes a DSPI instance for master mode operation to work with EDMA.
- `dspi_status_t 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` *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.

DSPI Master Driver

Non-blocking transfers

- `dspi_status_t DSPI_DRV_EdmaMasterTransfer` (uint32_t instance, const `dspi_edma_device_t *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 *frames-Transferred)
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.
- void `DSPI_DRV_EdmaMasterIRQHandler` (uint32_t instance)
Interrupt handler for DSPI master mode.

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.
- `dspi_status_t DSPI_DRV_MasterDeinit` (uint32_t instance)
Shuts down a DSPI instance.
- `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 *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 *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 *frames-Transferred)
Returns whether the previous transfer is completed.

- [dspi_status_t DSPI_DRV_MasterAbortTransfer](#) (uint32_t instance)
Terminates an asynchronous transfer early.
- void [DSPI_DRV_MasterIRQHandler](#) (uint32_t instance)
Interrupt handler for DSPI master mode.

11.3.9 Data Structure Documentation

11.3.9.1 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.

11.3.9.1.0.11 Field Documentation

11.3.9.1.0.11.1 uint32_t dspi_edma_device_t::bitsPerSec

11.3.9.2 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 [dspiSourceClock](#)
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](#)

DSPI Master Driver

- [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 * rxBuffer](#)
The buffer into which received bytes are placed.
- [uint32_t rxTransferByteCnt](#)
Number of bytes to receive.

11.3.9.2.0.12 Field Documentation

11.3.9.2.0.12.1 volatile bool [dsppi_edma_master_state_t::isTransferInProgress](#)

11.3.9.2.0.12.2 volatile bool [dsppi_edma_master_state_t::isTransferBlocking](#)

11.3.9.2.0.12.3 semaphore_t [dsppi_edma_master_state_t::irqSync](#)

11.3.9.2.0.12.4 [uint8_t*](#) [dsppi_edma_master_state_t::rxBuffer](#)

11.3.9.2.0.12.5 [uint32_t](#) [dsppi_edma_master_state_t::rxTransferByteCnt](#)

11.3.9.3 struct [dsppi_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

- [dsppi_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.
- [dsppi_which_pcs_config_t whichPcs](#)
Desired Peripheral Chip Select (pcs)
- [dsppi_pcs_polarity_config_t pcsPolarity](#)
Peripheral Chip Select (pcs) polarity setting.

11.3.9.3.0.13 Field Documentation

11.3.9.3.0.13.1 `dspi_pcs_polarity_config_t dspi_edma_master_user_config_t::pcsPolarity`

11.3.9.4 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.

11.3.9.4.0.14 Field Documentation

11.3.9.4.0.14.1 `uint32_t dspi_device_t::bitsPerSec`

11.3.9.5 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 * sendBuffer`
The buffer from which transmitted bytes are taken.
- `uint8_t * 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`

DSPI Master Driver

- *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.

11.3.9.5.0.15 Field Documentation

11.3.9.5.0.15.1 **volatile bool dspi_master_state_t::isTransferInProgress**

11.3.9.5.0.15.2 **const uint8_t* dspi_master_state_t::sendBuffer**

11.3.9.5.0.15.3 **uint8_t* dspi_master_state_t::receiveBuffer**

11.3.9.5.0.15.4 **volatile size_t dspi_master_state_t::remainingSendByteCount**

11.3.9.5.0.15.5 **volatile size_t dspi_master_state_t::remainingReceiveByteCount**

11.3.9.5.0.15.6 **volatile bool dspi_master_state_t::isTransferBlocking**

11.3.9.5.0.15.7 **semaphore_t dspi_master_state_t::irqSync**

11.3.9.6 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.

11.3.10 Function Documentation

11.3.10.1 `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)`

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>dspiEdmaState</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>dspi_edma_master_user_config_t</code> user configuration structure. The user populates the members of this structure and passes the pointer of the structure into the function.
<i>stcdSrc2Cmd-DataLast</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 <code>#pragma</code> directive to align a variable to a desired boundary.

Returns

An error code or `kStatus_DSPI_Success`.

DSPI Master Driver

11.3.10.2 `dspi_status_t 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

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

Returns

kStatus_DSPI_Success indicating successful de-initialization

11.3.10.3 dspi_status_t DSPI_DRV_EdmaMasterSetDelay (uint32_t instance, dspi_delay_type_t whichDelay, uint32_t delayInNanoSec, uint32_t * calculatedDelay)

This function uses 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 dspi_delay_type_t
<i>delayInNanoSec</i>	The desired delay value in nanoseconds.

DSPI Master Driver

<i>calculated-Delay</i>	The calculated delay that best matches the desired delay (in nanoseconds).
-------------------------	--

Returns

Either `kStatus_DSPI_Success` or `kStatus_DSPI_OutOfRange` if the desired delay exceeds the capability of the device.

11.3.10.4 `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`.

11.3.10.5 `dspi_status_t DSPI_DRV_EdmaMasterTransferBlocking (uint32_t instance,
const dspi_edma_device_t * 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.

DSPI Master Driver

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>transferByte-Count</i>	The number of bytes to send and receive.
<i>timeout</i>	A timeout for the transfer in milliseconds. 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.

11.3.10.6 dspi_status_t DSPI_DRV_EdmaMasterTransfer (uint32_t *instance*, const dspi_edma_device_t * *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.

<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.

11.3.10.7 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.

11.3.10.8 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.

DSPI Master Driver

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.

11.3.10.9 void DSPI_DRV_EdmaMasterIRQHandler (uint32_t *instance*)

This handler uses the buffers stored in the [dspimaster_state_t](#) structs to transfer data.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

11.3.10.10 dspimaster_status_t DSPI_DRV_MasterInit (uint32_t *instance*, dspimaster_state_t * *dspiState*, const dspimaster_user_config_t * *userConfig*)

This function uses a CPU interrupt driven method for transferring data. This function initializes the runtime state structure to track the ongoing transfers, un-gates 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 [dspimaster_state_t](#) and the [dspimaster_user_config_t](#) parameters and to call the DSPI_DRV_MasterInit function by passing in these parameters:

```

dspimaster_state_t dspimasterState; <- the user allocates memory for this structure
uint32_t calculatedBaudRate;
dspimaster_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.
<i>userConfig</i>	The dspi_master_user_config_t 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`.

11.3.10.11 dspi_status_t 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.
-----------------	---

Returns

`kStatus_DSPI_Success` indicating successful de-initialization

11.3.10.12 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.

DSPI Master Driver

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>delayInNanoSec</i>	The desired delay value in nanoseconds.
<i>calculated-Delay</i>	The calculated delay that best matches the desired delay (in nanoseconds).

Returns

Either `kStatus_DSPI_Success` or `kStatus_DSPI_OutOfRange` if the desired delay exceeds the capability of the device.

11.3.10.13 `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.
<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`.

11.3.10.14 `dspi_status_t DSPI_DRV_MasterTransferBlocking (uint32_t instance, const dspi_device_t * 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 <code>DSPI_DRV_MasterConfigureBus</code> 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.

DSPI Master Driver

<i>timeout</i>	A timeout for the transfer in milliseconds. 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.

11.3.10.15 `dspi_status_t DSPI_DRV_MasterTransfer (uint32_t instance, const dspi_device_t * device, const uint8_t * sendBuffer, uint8_t * receiveBuffer, size_t transferByteCount)`

This function returns immediately. The user needs to check whether the transfer is complete using the `DSPI_DRV_MasterGetTransferStatus` function. This function simultaneously sends and receives data on the SPI bus because the 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 <code>DSPI_DRV_MasterConfigureBus</code> 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.

11.3.10.16 `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.
<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.

11.3.10.17 dsp_i_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.

11.3.10.18 void DSPI_DRV_MasterIRQHandler (uint32_t instance)

This handler uses the buffers stored in the [dsp_i_master_state_t](#) structs to transfer data.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

11.3.11 Variable Documentation

11.3.11.1 SPI_Type* const g_dspiBase[SPI_INSTANCE_COUNT]

11.3.11.2 const IRQn_Type g_dspilrqld[SPI_INSTANCE_COUNT]

11.3.11.3 SPI_Type* const g_dspiBase[SPI_INSTANCE_COUNT]

11.3.11.4 const IRQn_Type g_dspilrqld[SPI_INSTANCE_COUNT]

DSPI Slave Driver

11.4 DSPI Slave Driver

11.4.1 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. In the subsequent chapters, the enhanced DMA-driven driver is referred to 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. In the subsequent chapters, either set of drivers is referred to as the "DSPI slave driver" when discussing items that pertain to either driver. Note, when using the DMA driven DSPI driver, eDMA module needs to be initialized. An example is shown in the Initialization chapter.

This is a step-by-step overview to set up the DSPI for SPI slave mode operations. For API specific examples, see the examples below. This example 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 because the SPI interrupt handler needs to be changed. The interrupt driver calls the [DSPI_DRV_IRQ_Handler\(\)](#) function and the eDMA driver calls the [DSPI_DRV_EdmaIRQHandler\(\)](#) function. See 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, separately set up and initialize another instance for DMA operations. The user can also de-initialize the current interrupt-driven DSPI instance and re-initialize it for DMA operations. Note, that, because the DMA-driven driver also uses interrupts, direct the IRQ handler from the vector table to the desired driver's IRQ handler. See files `fsl_dspi_irq.c` and `fsl_dspi_edma_irq.c` for examples to re-direct the IRQ handlers from the vector table to the interrupt-driven and DMA-driven driver IRQ handlers. These files need to be included in the applications project in order to direct the DSPI interrupt vectors to the proper IRQ handlers. The `fsl_dspi_shared_function.c` and `fsl_dspi_edma_shared_`

function.c files 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)`.

When using the DSPI driver with the eDMA, some DSPI instances do not support separate DMA requests for TX and RX channels. In those cases, with a single shared DMA request for TX and RX DMA channels, the driver links one channel to the other in order to take advantage of DMA transfers. However, the drawback to using an instance with shared DMA requests limits the maximum amount of data the driver can transfer. This limit depends on the bits/frame setting.

For DSPI instances with separate TX and RX DMA requests, the maximum number of bytes that can be transferred are: 8-bit setting: 32767 bytes 16-bit setting: 65534 bytes

For DSPI instances with a shared TX and RX DMA request, the maximum number of bytes that can be transferred are: 8-bit setting: 511 bytes 16-bit setting: 1022 bytes

See the microcontroller-specific documentation to see which DSPI instance have separate or shared TX and RX DMA requests. Additionally, see the microcontroller-specific feature header file to see which DSPI instance have separate or shared TX and RX DMA requests.

11.4.2 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 the `dspi_slave_state_t`. The state structure for the DMA driven driver is called the `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. The DSPI slave driver fills out the members.

11.4.3 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 the `dspi_slave_user_config_t`. The user configuration structure for the DMA driven driver is called the `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.

11.4.4 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 the `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:

DSPI Slave Driver

```

// 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, &dspiSlaveState, &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);

dspi_edma_slave_state_t dspiEdmaSlaveState;
// update configs
dspi_edma_slave_user_config_t slaveEdmaUserConfig;
slaveEdmaUserConfig.dataConfig.clkPhase =
    kDspiClockPhase_FirstEdge;
slaveEdmaUserConfig.dataConfig.clkPolarity =
    kDspiClockPolarity_ActiveHigh;
slaveEdmaUserConfig.dataConfig.bitsPerFrame = bitCount;
slaveEdmaUserConfig.dummyPattern = DSPI_DEFAULT_DUMMY_PATTERN;

// init the slave (DMA driven)
DSPI_DRV_EdmaSlaveInit(instance, &dspiEdmaSlaveState, &slaveEdmaUserConfig);

```

11.4.5 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 completed.

This is an example of blocking and non-blocking call (interrupt driven):

```

dspi_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 this example:

```
dspi_status_t result;
// 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);
```

11.4.6 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 [dspi_edma_slave_user_config_t](#)
User configuration structure for the DSPI slave driver. [More...](#)
- struct [dspi_edma_slave_state_t](#)
Runtime state structure of the DSPI slave driver. [More...](#)
- struct [dspi_slave_user_config_t](#)
User configuration structure for the DSPI slave driver. [More...](#)
- struct [dspi_slave_state_t](#)
Runtime state of the DSPI slave driver. [More...](#)

DSPI Slave Driver

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

- SPI_Type *const `g_dspiBase` [SPI_INSTANCE_COUNT]
Table of base pointers for SPI instances.
- const IRQn_Type `g_dspiIrqId` [SPI_INSTANCE_COUNT]
Table to save DSPI IRQ enumeration numbers defined in the CMSIS header file.
- `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 * `dspi_edma_slave_state_t::sendBuffer`
Pointer to transmit buffer.
- uint8_t * `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.
- SPI_Type *const `g_dspiBase` [SPI_INSTANCE_COUNT]
Table of base pointers for SPI instances.
- const IRQn_Type `g_dspiIrqId` [SPI_INSTANCE_COUNT]
Table to save DSPI IRQ enumeration numbers defined in the CMSIS header file.

Initialization and shutdown

- [dspi_status_t DSPI_DRV_EdmaSlaveInit](#) (uint32_t instance, [dspi_edma_slave_state_t](#) *dspiState, const [dspi_edma_slave_user_config_t](#) *slaveConfig)
Initializes a DSPI instance for a slave mode operation, using eDMA mechanism.
- [dspi_status_t 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 the SPI bus using the eDMA and blocking call.

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)
Aborts the transfer that started by a non-blocking call to the eDMA transfer function.
- [dspi_status_t DSPI_DRV_EdmaSlaveGetTransferStatus](#) (uint32_t instance, uint32_t *framesTransferred)
Returns whether the previous transfer is finished.
- void [DSPI_DRV_EdmaSlaveIRQHandler](#) (uint32_t instance)
DSPI slave interrupt handler.

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.
- [dspi_status_t 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.

DSPI Slave Driver

Non-blocking transfers

- [dsp_i_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 a non-blocking call.
- [dsp_i_status_t DSPI_DRV_SlaveAbortTransfer](#) (uint32_t instance)
Aborts the transfer that started by a non-blocking call to a transfer function.
- [dsp_i_status_t DSPI_DRV_SlaveGetTransferStatus](#) (uint32_t instance, uint32_t *framesTransferred)
Returns whether the previous transfer is finished.
- void [DSPI_DRV_SlaveIRQHandler](#) (uint32_t instance)
DSPI Slave Generic IRQ handler.

11.4.7 Data Structure Documentation

11.4.7.1 struct dsp_i_edma_slave_user_config_t

Data Fields

- [dsp_i_data_format_config_t dataConfig](#)
Data format configuration structure.
- uint16_t [dummyPattern](#)
Dummy data value.

11.4.7.2 struct dsp_i_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.
- [dsp_i_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 * [sendBuffer](#)
Pointer to transmit buffer.
- uint8_t * [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.

11.4.7.3 struct dspi_slave_user_config_t

Data Fields

- dspi_data_format_config_t **dataConfig**
Data format configuration structure.
- uint16_t **dummyPattern**
Dummy data value.

11.4.7.4 struct dspi_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 * **sendBuffer**
Pointer to transmit buffer.
- uint8_t * **receiveBuffer**
Pointer to receive buffer.
- volatile int32_t **remainingSendByteCount**
Number of bytes remaining to send.

DSPI Slave Driver

- volatile int32_t [remainingReceiveByteCount](#)
Number of bytes remaining to receive.
- bool [isSync](#)
Indicates the function call is sync or async.

11.4.7.4.0.16 Field Documentation

11.4.7.4.0.16.1 volatile bool [dsppi_slave_state_t::isTransferInProgress](#)

11.4.7.4.0.16.2 volatile int32_t [dsppi_slave_state_t::remainingSendByteCount](#)

11.4.7.4.0.16.3 volatile int32_t [dsppi_slave_state_t::remainingReceiveByteCount](#)

11.4.8 Function Documentation

11.4.8.1 [dsppi_status_t DSPI_DRV_EdmaSlaveInit \(uint32_t *instance*, \[dsppi_edma_slave_state_t\]\(#\) * *dsppiState*, const \[dsppi_edma_slave_user_config_t\]\(#\) * *slaveConfig* \)](#)

This function un-gates the clock to the DSPI module, initializes the DSPI for slave mode and initializes eDMA channels. After it is 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 uses the eDMA mechanism.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
<i>dsppiState</i>	The pointer to the DSPI slave driver state structure.
<i>slaveConfig</i>	The configuration structure dsppi_edma_slave_user_config_t which configures the data bus format.

Returns

An error code or [kStatus_DSPI_Success](#).

11.4.8.2 [dsppi_status_t DSPI_DRV_EdmaSlaveDeinit \(uint32_t *instance* \)](#)

Disables the DSPI module, gates its clock, changes the DSPI slave driver state to NonInit for DSPI slave module which is initialized with the eDMA mechanism. After de-initialization, the user can re-initialize the DSPI slave module with other mechanisms.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

Returns

kStatus_DSPI_Success indicating successful de-initialization or error code

11.4.8.3 dspi_status_t DSPI_DRV_EdmaSlaveTransferBlocking (uint32_t instance, const uint8_t * sendBuffer, uint8_t * receiveBuffer, uint32_t transferByteCount, uint32_t timeOut)

This function checks driver status, mechanism and transmits/receives data through SPI bus. If sendBuffer is NULL, transmit process is ignored, and if the receiveBuffer is NULL, the receive process is ignored. If both the receiveBuffer and the sendBuffer are available, the transmit and receive progress is processed. If only the receiveBuffer is available, the receive is processed, Otherwise, the transmit is processed. This function only returns when its processes are complete. This function uses the 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 milliseconds 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.

11.4.8.4 dspi_status_t DSPI_DRV_EdmaSlaveTransfer (uint32_t instance, const uint8_t * sendBuffer, uint8_t * receiveBuffer, uint32_t transferByteCount)

This function checks the driver status then sets buffer pointers to receive and transmit SPI data. If the send-Buffer is NULL, transmit process is ignored. If the receiveBuffer is NULL, the receive process is ignored. If both the receiveBuffer and sendBuffer are available, the transfer is done when the kDspiTxDone and the kDspiRxDone are set. If only the receiveBuffer is available, the transfer is done when the kDspiRxDone flag is set. Otherwise, the transfer is done when the kDspiTxDone is set. This function uses the eDMA mechanism.

DSPI Slave Driver

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.

11.4.8.5 `dsapi_status_t DSPI_DRV_EdmaSlaveAbortTransfer (uint32_t instance)`

This function stops the transfer which was started by the [DSPI_DRV_EdmaSlaveTransfer\(\)](#) function.

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.

11.4.8.6 `dsapi_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.

11.4.8.7 void DSPI_DRV_EdmaSlaveIRQHandler (uint32_t instance)

This function is DSPI slave interrupt handler using eDMA mechanism. The pupose of this interrupt handler is indicates when the transfer is really finished. The eDMA only used to copy data from/to RX FIFO/TX FIFO, but it not sure the data was transmitted to the master. So must have to enable this interrupt to do it. This interrupt only be enabled when the last four FIFO will be transmitted.

Parameters

<i>instance</i>	The SPI number
-----------------	----------------

11.4.8.8 dsp_i_status_t DSPI_DRV_SlaveInit (uint32_t instance, dsp_i_slave_state_t * dsp_iState, const dsp_i_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 dsp_i_slave_user_config_t which configures the data bus format.

Returns

An error code or kStatus_DSPI_Success.

DSPI Slave Driver

11.4.8.9 `dspi_status_t DSPI_DRV_SlaveDeinit (uint32_t instance)`

Disables the 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.
-----------------	---

Returns

`kStatus_DSPI_Success` indicating successful de-initialization or error code

11.4.8.10 `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, the transmit process is ignored, and if `receiveBuffer` is NULL the receive process is ignored. If both `receiveBuffer` and `sendBuffer` are available, both the transmit and the receive progress are processed. If only the `receiveBuffer` is available, the receive is processed. Otherwise, the transmit is processed. This function only returns when its processes are complete. This function uses an 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.
<i>timeout</i>	The maximum number of milliseconds 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.

11.4.8.11 dspi_status_t DSPI_DRV_SlaveTransfer (uint32_t instance, const uint8_t * sendBuffer, uint8_t * receiveBuffer, uint32_t transferByteCount)

This function checks the driver status and sets buffer pointers to receive and transmit the SPI data. If the sendBuffer is NULL, the transmit process is ignored. If the receiveBuffer is NULL, the receive process is ignored. If both the receiveBuffer and the sendBuffer are , available the transfer is done when the kDspi-TxDone and the kDspiRxDone are set. If only the receiveBuffer is available, the transfer is done when the kDspiRxDone flag is set. Otherwise, the transfer is done when the kDspiTxDone is set. This function uses an 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.

11.4.8.12 dspi_status_t DSPI_DRV_SlaveAbortTransfer (uint32_t instance)

This function stops the transfer which started by calling the [DSPI_DRV_SlaveTransfer\(\)](#) function.

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.

11.4.8.13 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.

DSPI Slave Driver

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.

11.4.8.14 void DSPI_DRV_SlaveIRQHandler (uint32_t *instance*)

This handler check errors of driver and it puts data into Tx FIFO, gets data from Rx FIFO whenever data transmitting/received.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

11.4.9 Variable Documentation

11.4.9.1 SPI_Type* const g_dsipiBase[SPI_INSTANCE_COUNT]

11.4.9.2 const IRQn_Type g_dspilrqld[SPI_INSTANCE_COUNT]

11.4.9.3 volatile bool dspi_edma_slave_state_t::isTransferInProgress

11.4.9.4 volatile int32_t dspi_edma_slave_state_t::remainingSendByteCount

11.4.9.5 volatile int32_t dspi_edma_slave_state_t::remainingReceiveByteCount

11.4.9.6 SPI_Type* const g_dsipiBase[SPI_INSTANCE_COUNT]

11.4.9.7 const IRQn_Type g_dspilrqld[SPI_INSTANCE_COUNT]

11.5 DSPI Shared IRQ Driver

11.5.1 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.

Variables

- SPI_Type *const [g_dspiBase](#) [SPI_INSTANCE_COUNT]
Table of base pointers for SPI instances.
- const IRQn_Type [g_dspiIrqId](#) [SPI_INSTANCE_COUNT]
Table to save DSPI IRQ enumeration numbers defined in the CMSIS header file.
- SPI_Type *const [g_dspiBase](#) [SPI_INSTANCE_COUNT]
Table of base pointers for SPI instances.
- const IRQn_Type [g_dspiIrqId](#) [SPI_INSTANCE_COUNT]
Table to save DSPI IRQ enumeration numbers defined in the CMSIS header file.

11.5.2 Function Documentation

11.5.2.1 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.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

11.5.2.2 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

DSPI Shared IRQ Driver

marked as weak). This would actually be a program error, because it means the master/slave configuration for the IRQ was set incorrectly.

Parameters

<i>instance</i>	The instance number of the DSPI peripheral.
-----------------	---

11.5.3 Variable Documentation

11.5.3.1 `SPI_Type* const g_dspiBase[SPI_INSTANCE_COUNT]`

11.5.3.2 `const IRQn_Type g_dspilrqld[SPI_INSTANCE_COUNT]`

11.5.3.3 `SPI_Type* const g_dspiBase[SPI_INSTANCE_COUNT]`

11.5.3.4 `const IRQn_Type g_dspilrqld[SPI_INSTANCE_COUNT]`

Chapter 12

Enhanced Direct Memory Access (eDMA)

12.1 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

12.2 eDMA HAL driver

12.2.1 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...](#)
- struct [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](#) (DMA_Type *base)
Initializes eDMA module to known state.
- void [EDMA_HAL_CancelTransfer](#) (DMA_Type *base)

- Cancels the remaining data transfer.*

 - void [EDMA_HAL_ErrorCancelTransfer](#) (DMA_Type *base)
- Cancels the remaining data transfer and treats it as an error condition.*

 - static void [EDMA_HAL_SetHaltCmd](#) (DMA_Type *base, bool halt)
- Halts/Un-halts the DMA Operations.*

 - static void [EDMA_HAL_SetHaltOnErrorCmd](#) (DMA_Type *base, bool haltOnError)
- Halts or does not halt the eDMA module when an error occurs.*

 - static void [EDMA_HAL_SetDebugCmd](#) (DMA_Type *base, bool enable)
- Enables/Disables the eDMA DEBUG mode.*

eDMA HAL driver channel priority and arbitration configuration.

- static void [EDMA_HAL_SetChannelPreemptMode](#) (DMA_Type *base, uint32_t channel, bool preempt, bool preemption)

Sets the preempt and preemption feature for the eDMA channel.
- static void [EDMA_HAL_SetChannelPriority](#) (DMA_Type *base, uint32_t channel, [edma_channel_priority_t](#) priority)

Sets the eDMA channel priority.
- static void [EDMA_HAL_SetChannelArbitrationMode](#) (DMA_Type *base, [edma_channel_arbitration_t](#) channelArbitration)

Sets the channel arbitration algorithm.

eDMA HAL driver configuration and operation.

- static void [EDMA_HAL_SetMinorLoopMappingCmd](#) (DMA_Type *base, bool enable)

Enables/Disables the minor loop mapping.
- static void [EDMA_HAL_SetContinuousLinkCmd](#) (DMA_Type *base, bool continuous)

Enables or disables the continuous transfer mode.
- [edma_error_status_all_t](#) [EDMA_HAL_GetErrorStatus](#) (DMA_Type *base)

Gets the error status of the eDMA module.
- void [EDMA_HAL_SetErrorIntCmd](#) (DMA_Type *base, bool enable, [edma_channel_indicator_t](#) channel)

Enables/Disables the error interrupt for channels.
- static uint32_t [EDMA_HAL_GetErrorIntStatusFlag](#) (DMA_Type *base)

Gets the eDMA error interrupt status.
- static void [EDMA_HAL_ClearErrorIntStatusFlag](#) (DMA_Type *base, [edma_channel_indicator_t](#) channel)

Clears the error interrupt status for the eDMA channel or channels.
- void [EDMA_HAL_SetDmaRequestCmd](#) (DMA_Type *base, [edma_channel_indicator_t](#) channel, bool enable)

Enables/Disables the DMA request for the channel or all channels.
- static bool [EDMA_HAL_GetDmaRequestStatusFlag](#) (DMA_Type *base, uint32_t channel)

Gets the eDMA channel DMA request status.
- static void [EDMA_HAL_ClearDoneStatusFlag](#) (DMA_Type *base, [edma_channel_indicator_t](#) channel)

Clears the done status for a channel or all channels.

eDMA HAL driver

- static void [EDMA_HAL_TriggerChannelStart](#) (DMA_Type *base, [edma_channel_indicator_t](#) channel)
Triggers the eDMA channel.
- static bool [EDMA_HAL_GetIntStatusFlag](#) (DMA_Type *base, uint32_t channel)
Gets the eDMA channel interrupt request status.
- static void [EDMA_HAL_ClearIntStatusFlag](#) (DMA_Type *base, [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](#) (DMA_Type *base, uint32_t channel)
Clears all registers to 0 for the hardware TCD.
- static void [EDMA_HAL_HTCDSrcAddr](#) (DMA_Type *base, uint32_t channel, uint32_t address)
Configures the source address for the hardware TCD.
- static void [EDMA_HAL_HTCDSrcOffset](#) (DMA_Type *base, uint32_t channel, int16_t offset)
Configures the source address signed offset for the hardware TCD.
- void [EDMA_HAL_HTCDSrcAttribute](#) (DMA_Type *base, 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_HTCDSrcNbytes](#) (DMA_Type *base, uint32_t channel, uint32_t nbytes)
Configures the nbytes for the eDMA channel.
- uint32_t [EDMA_HAL_HTCDSrcNbytes](#) (DMA_Type *base, uint32_t channel)
Gets the nbytes configuration data for the hardware TCD.
- void [EDMA_HAL_HTCDSrcMinorLoopOffset](#) (DMA_Type *base, uint32_t channel, [edma_miniorloop_offset_config_t](#) *config)
Configures the minor loop offset for the hardware TCD.
- static void [EDMA_HAL_HTCDSrcLastAdjust](#) (DMA_Type *base, uint32_t channel, int32_t size)
Configures the last source address adjustment for the hardware TCD.
- static void [EDMA_HAL_HTCDSrcDestAddr](#) (DMA_Type *base, uint32_t channel, uint32_t address)
Configures the destination address for the hardware TCD.
- static void [EDMA_HAL_HTCDSrcDestOffset](#) (DMA_Type *base, uint32_t channel, int16_t offset)
Configures the destination address signed offset for the hardware TCD.
- static void [EDMA_HAL_HTCDSrcDestLastAdjust](#) (DMA_Type *base, uint32_t channel, uint32_t adjust)
Configures the last source address adjustment.
- void [EDMA_HAL_HTCDSrcScatterGatherLink](#) (DMA_Type *base, 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_HTCDSrcBandwidth](#) (DMA_Type *base, uint32_t channel, [edma_bandwidth_config_t](#) bandwidth)
Configures the bandwidth for the hardware TCD.
- static void [EDMA_HAL_HTCDSrcChannelMajorLink](#) (DMA_Type *base, uint32_t channel, uint32_t majorChannel, bool enable)

- Configures the major channel link the hardware TCD.*

 - static void [EDMA_HAL_HTCDSetScatterGatherCmd](#) (DMA_Type *base, uint32_t channel, bool enable)

Enables/Disables the scatter/gather feature for the hardware TCD.

 - static void [EDMA_HAL_HTCDSetDisableDmaRequestAfterTCDDoneCmd](#) (DMA_Type *base, uint32_t channel, bool disable)

Disables/Enables the DMA request after the major loop completes for the hardware TCD.

 - static void [EDMA_HAL_HTCDSetHalfCompleteIntCmd](#) (DMA_Type *base, uint32_t channel, bool enable)

Enables/Disables the half complete interrupt for the hardware TCD.

 - static void [EDMA_HAL_HTCDSetIntCmd](#) (DMA_Type *base, uint32_t channel, bool enable)

Enables/Disables the interrupt after the major loop completes for the hardware TCD.

 - static void [EDMA_HAL_HTCDTriggerChannelStart](#) (DMA_Type *base, uint32_t channel)

Triggers the start bits for the hardware TCD.

 - static bool [EDMA_HAL_HTCDGetChannelActiveStatus](#) (DMA_Type *base, uint32_t channel)

Checks whether the channel is running for the hardware TCD.

 - void [EDMA_HAL_HTCDSetChannelMinorLink](#) (DMA_Type *base, uint32_t channel, uint32_t linkChannel, bool enable)

Sets the channel minor link for the hardware TCD.

 - void [EDMA_HAL_HTCDSetMajorCount](#) (DMA_Type *base, uint32_t channel, uint32_t count)

Sets the major iteration count according to minor loop channel link setting.

 - uint32_t [EDMA_HAL_HTCDGetFinishedBytes](#) (DMA_Type *base, uint32_t channel)

Gets the number of bytes already transferred for the hardware TCD.

 - uint32_t [EDMA_HAL_HTCDGetUnfinishedBytes](#) (DMA_Type *base, uint32_t channel)

Gets the number of bytes haven't transferred for the hardware TCD.

 - static bool [EDMA_HAL_HTCDGetDoneStatusFlag](#) (DMA_Type *base, 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)
- 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](#) (DMA_Type *base, edma_software_tcd_t *stcd, uint32_t nbytes)
- Configures the nbytes for software TCD.*
- void [EDMA_HAL_STCDSetMinorLoopOffset](#) (DMA_Type *base, 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)

eDMA HAL driver

- 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)

Disables/Enables 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](#) (DMA_Type *base, uint32_t channel, edma_software_tcd_t *stcd)

Copy the software TCD configuration to the hardware TCD.

 - [edma_status_t](#) [EDMA_HAL_STCDSetBasicTransfer](#) (DMA_Type *base, edma_software_tcd_t *stcd, edma_transfer_config_t *config, bool enableInt, bool disableDmaRequest)

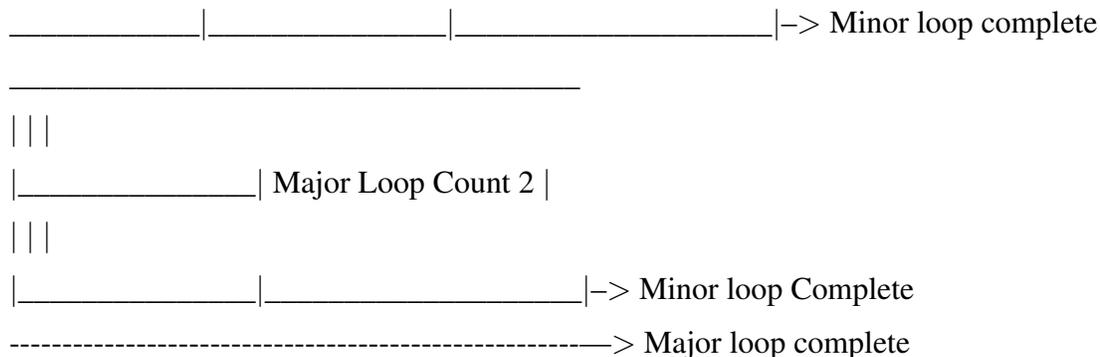
Set the basic transfer for software TCD.

12.2.2 Data Structure Documentation

12.2.2.1 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

12.2.2.1.0.17 Field Documentation

12.2.2.1.0.17.1 `uint32_t edma_transfer_config_t::srcAddr`

12.2.2.1.0.17.2 `uint32_t edma_transfer_config_t::destAddr`

12.2.2.1.0.17.3 `edma_transfer_size_t edma_transfer_config_t::srcTransferSize`

12.2.2.1.0.17.4 `edma_transfer_size_t edma_transfer_config_t::destTransferSize`

12.2.2.1.0.17.5 `int16_t edma_transfer_config_t::srcOffset`

12.2.2.1.0.17.6 `uint32_t edma_transfer_config_t::srcLastAddrAdjust`

12.2.2.1.0.17.7 `uint32_t edma_transfer_config_t::destLastAddrAdjust`

Note here it is only valid when scatter/gather feature is not enabled.

12.2.2.1.0.17.8 `edma_modulo_t edma_transfer_config_t::srcModulo`

12.2.2.1.0.17.9 `edma_modulo_t edma_transfer_config_t::destModulo`

12.2.2.1.0.17.10 `uint32_t edma_transfer_config_t::minorLoopCount`

Number of bytes to be transferred in each service request of the channel.

12.2.2.1.0.17.11 `uint16_t edma_transfer_config_t::majorLoopCount`

12.2.2.2 `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.

12.2.2.2.0.18 Field Documentation

12.2.2.2.0.18.1 bool `edma_minorloop_offset_config_t::enableSrcMinorloop`

12.2.2.2.0.18.2 bool `edma_minorloop_offset_config_t::enableDestMinorloop`

12.2.2.2.0.18.3 uint32_t `edma_minorloop_offset_config_t::offset`

12.2.2.3 struct `edma_error_status_all_t`

Data Fields

- uint8_t `errorChannel`
Error channel number of the cancelled channel number.
- bool `destinationBusError`
Bus error on destination address.
- bool `sourceBusError`
Bus error on the SRC address.
- bool `scatterOrGatherConfigurationError`
Error on the Scatter/Gather address.
- bool `nbyteOrCiterConfigurationError`
NBYTES/CITER configuration error.
- bool `destinationOffsetError`
Destination offset error.
- bool `destinationAddressError`
Destination address error.
- bool `sourceOffsetError`
Source offset error.
- bool `sourceAddressError`
Source address error.
- bool `channelPriorityError`
Channel priority error.
- bool `transferCancelledError`
Transfer cancelled.
- bool `orOfAllError`
Logical OR all ERR status bits.

12.2.2.4 struct `edma_software_tcd_t`

12.2.3 Enumeration Type Documentation

12.2.3.1 enum `edma_status_t`

Enumerator

kStatus_EDMA_InvalidArgument Parameter is invalid.

kStatus_EDMA_Fail Failed operation.

eDMA HAL driver

12.2.3.2 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.

12.2.3.3 enum edma_channel_indicator_t

Enumerator

kEDMAChannel0 Channel 0.

12.2.3.4 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.

12.2.4 Function Documentation

12.2.4.1 void EDMA_HAL_Init (DMA_Type * base)

Parameters

<i>base</i>	Register base address for eDMA module.
-------------	--

12.2.4.2 void EDMA_HAL_CancelTransfer (DMA_Type * base)

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>base</i>	Register base address for eDMA module.
-------------	--

12.2.4.3 void EDMA_HAL_ErrorCancelTransfer (DMA_Type * *base*)

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>base</i>	Register base address for eDMA module.
-------------	--

12.2.4.4 static void EDMA_HAL_SetHaltCmd (DMA_Type * *base*, bool *halt*) [inline], [static]

This function stalls/un-stalls the start of any new channels. Executing channels are allowed to be completed.

Parameters

<i>base</i>	Register base address for eDMA module.
<i>halt</i>	Halts (true) or un-halts (false) eDMA transfer.

12.2.4.5 static void EDMA_HAL_SetHaltOnErrorCmd (DMA_Type * *base*, 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.

Parameters

<i>base</i>	Register base address for eDMA module.
<i>haltOnError</i>	Halts (true) or not halt (false) eDMA module when an error occurs.

12.2.4.6 static void EDMA_HAL_SetDebugCmd (DMA_Type * *base*, 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

eDMA HAL driver

system exits debug mode or when the EDBG bit is cleared.

Parameters

<i>base</i>	Register base address for eDMA module.
<i>enable</i>	Enables (true) or Disable (false) eDMA module debug mode.

12.2.4.7 static void EDMA_HAL_SetChannelPreemptMode (DMA_Type * *base*, uint32_t *channel*, bool *preempt*, bool *preemption*) [inline], [static]

This function sets the preempt and preemption features.

Parameters

<i>base</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).

12.2.4.8 static void EDMA_HAL_SetChannelPriority (DMA_Type * *base*, uint32_t *channel*, edma_channel_priority_t *priority*) [inline], [static]

Parameters

<i>base</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.

12.2.4.9 static void EDMA_HAL_SetChannelArbitrationMode (DMA_Type * *base*, edma_channel_arbitration_t *channelArbitration*) [inline], [static]

Parameters

eDMA HAL driver

<i>base</i>	Register base address for eDMA module.
<i>channel-Arbitration</i>	Round-Robin way for fixed priority way.

12.2.4.10 static void EDMA_HAL_SetMinorLoopMappingCmd (DMA_Type * *base*, 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>base</i>	Register base address for eDMA module.
<i>enable</i>	Enables (true) or Disable (false) minor loop mapping.

12.2.4.11 static void EDMA_HAL_SetContinuousLinkCmd (DMA_Type * *base*, 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.

Parameters

<i>base</i>	Register base address for eDMA module.
<i>continuous</i>	Enables (true) or Disable (false) continuous transfer mode.

12.2.4.12 edma_error_status_all_t EDMA_HAL_GetErrorStatus (DMA_Type * *base*)

Parameters

<i>base</i>	Register base address for eDMA module.
-------------	--

Returns

Detailed information of the error type in the eDMA module.

12.2.4.13 void EDMA_HAL_SetErrorIntCmd (DMA_Type * *base*, bool *enable*,
edma_channel_indicator_t *channel*)

eDMA HAL driver

Parameters

<i>base</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.

12.2.4.14 static uint32_t EDMA_HAL_GetErrorIntStatusFlag (DMA_Type * *base*) [inline], [static]

Parameters

<i>base</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'.

12.2.4.15 static void EDMA_HAL_ClearErrorIntStatusFlag (DMA_Type * *base*, edma_channel_indicator_t *channel*) [inline], [static]

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	Channel indicator. If kEDMAAllChannel is selected, all channels' error interrupt status will be cleared.

12.2.4.16 void EDMA_HAL_SetDmaRequestCmd (DMA_Type * *base*, edma_channel_indicator_t *channel*, bool *enable*)

Parameters

<i>base</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.

12.2.4.17 static bool EDMA_HAL_GetDmaRequestStatusFlag (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

<i>base</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).

12.2.4.18 static void EDMA_HAL_ClearDoneStatusFlag (DMA_Type * *base*, edma_channel_indicator_t *channel*) [inline], [static]

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	Channel indicator. If kEDMAAllChannel is selected, all channels' done status will be cleared.

12.2.4.19 static void EDMA_HAL_TriggerChannelStart (DMA_Type * *base*, edma_channel_indicator_t *channel*) [inline], [static]

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	Channel indicator. If kEDMAAllChannel is selected, all channels are triggered.

12.2.4.20 static bool EDMA_HAL_GetIntStatusFlag (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

eDMA HAL driver

Parameters

<i>base</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).

12.2.4.21 static void EDMA_HAL_ClearIntStatusFlag (DMA_Type * *base*, edma_channel_indicator_t *channel*) [inline], [static]

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	Channel indicator. If kEDMAAllChannel is selected, all channels' interrupt status will be cleared.

12.2.4.22 void EDMA_HAL_HTCDClearReg (DMA_Type * *base*, uint32_t *channel*)

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

12.2.4.23 static void EDMA_HAL_HTCDSetSrcAddr (DMA_Type * *base*, uint32_t *channel*, uint32_t *address*) [inline], [static]

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>address</i>	The pointer to the source memory address.

12.2.4.24 static void EDMA_HAL_HTCDSetSrcOffset (DMA_Type * *base*, 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>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>offset</i>	signed-offset for source address.

12.2.4.25 void EDMA_HAL_HTCDSetAttribute (DMA_Type * *base*, uint32_t *channel*, edma_modulo_t *srcModulo*, edma_modulo_t *destModulo*, edma_transfer_size_t *srcTransferSize*, edma_transfer_size_t *destTransferSize*)

Parameters

<i>base</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>destTransfer-Size</i>	Enum type for destination transfer size.

12.2.4.26 void EDMA_HAL_HTCDSetNbytes (DMA_Type * *base*, uint32_t *channel*, uint32_t *nbytes*)

Note here that user need firstly configure the minor loop mapping feature and then call this function.

Parameters

eDMA HAL driver

<i>base</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

12.2.4.27 `uint32_t EDMA_HAL_HTCDGetNbytes (DMA_Type * base, 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>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

nbytes configuration according to minor loop setting.

12.2.4.28 `void EDMA_HAL_HTCDSetMinorLoopOffset (DMA_Type * base, 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>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>config</i>	Configuration data structure for the minor loop offset

12.2.4.29 `static void EDMA_HAL_HTCDSetSrcLastAdjust (DMA_Type * base, 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>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>size</i>	adjustment value

12.2.4.30 static void EDMA_HAL_HTCDSetDestAddr (DMA_Type * *base*, uint32_t *channel*, uint32_t *address*) [inline], [static]

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>address</i>	The pointer to the destination address.

12.2.4.31 static void EDMA_HAL_HTCDSetDestOffset (DMA_Type * *base*, 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>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>offset</i>	signed-offset

12.2.4.32 static void EDMA_HAL_HTCDSetDestLastAdjust (DMA_Type * *base*, 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>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>adjust</i>	adjustment value

eDMA HAL driver

12.2.4.33 void EDMA_HAL_HTCDSetScatterGatherLink (DMA_Type * *base*, 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 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>base</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.

12.2.4.34 static void EDMA_HAL_HTCDSetBandwidth (DMA_Type * *base*, 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.

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>bandwidth</i>	enum type for bandwidth control

12.2.4.35 static void EDMA_HAL_HTCDSetChannelMajorLink (DMA_Type * *base*, 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>base</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.

12.2.4.36 `static void EDMA_HAL_HTCDSetScatterGatherCmd (DMA_Type * base,
uint32_t channel, bool enable) [inline], [static]`

eDMA HAL driver

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>enable</i>	Enables (true) /Disables (false) scatter/gather feature.

12.2.4.37 static void EDMA_HAL_HTCDSetDisableDmaRequestAfterTCDDoneCmd (DMA_Type * *base*, 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>base</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.

12.2.4.38 static void EDMA_HAL_HTCDSetHalfCompleteIntCmd (DMA_Type * *base*, 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.

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>enable</i>	Enable (true) /Disable (false) half complete interrupt.

12.2.4.39 static void EDMA_HAL_HTCDSetIntCmd (DMA_Type * *base*, 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>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>enable</i>	Enable (true) /Disable (false) interrupt after TCD done.

12.2.4.40 static void EDMA_HAL_HTCDTriggerChannelStart (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

The eDMA hardware automatically clears this flag after the channel begins execution.

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

12.2.4.41 static bool EDMA_HAL_HTCDGetChannelActiveStatus (DMA_Type * *base*, uint32_t *channel*) [inline], [static]

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

True stands for running. False stands for not.

12.2.4.42 void EDMA_HAL_HTCDSetChannelMinorLink (DMA_Type * *base*, uint32_t *channel*, uint32_t *linkChannel*, bool *enable*)

Parameters

<i>base</i>	Register base address for eDMA module.
-------------	--

eDMA HAL driver

<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.

12.2.4.43 void EDMA_HAL_HTCDSetMajorCount (DMA_Type * *base*, 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>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>count</i>	major loop count

12.2.4.44 uint32_t EDMA_HAL_HTCDGetFinishedBytes (DMA_Type * *base*, uint32_t *channel*)

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

data bytes already transferred

12.2.4.45 uint32_t EDMA_HAL_HTCDGetUnfinishedBytes (DMA_Type * *base*, uint32_t *channel*)

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

data bytes already transferred

12.2.4.46 `static bool EDMA_HAL_HTCDGetDoneStatusFlag (DMA_Type * base, uint32_t channel) [inline], [static]`

eDMA HAL driver

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.

Returns

If channel done.

12.2.4.47 `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.

12.2.4.48 `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.

12.2.4.49 `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>destTransfer-Size</i>	Enum type for destination transfer size.

12.2.4.50 void EDMA_HAL_STCDSetNbytes (DMA_Type * base, 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>base</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

12.2.4.51 void EDMA_HAL_STCDSetMinorLoopOffset (DMA_Type * base, 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>base</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

eDMA HAL driver

12.2.4.52 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

12.2.4.53 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.

12.2.4.54 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

12.2.4.55 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

12.2.4.56 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.

12.2.4.57 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.

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>bandwidth</i>	enum type for bandwidth control

12.2.4.58 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.

eDMA HAL driver

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.

12.2.4.59 `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.

12.2.4.60 `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.

12.2.4.61 `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.

12.2.4.62 `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.

eDMA HAL driver

Parameters

<i>stcd</i>	The pointer to the software TCD.
<i>enable</i>	Enable (true) /Disable (false) interrupt after TCD done.

12.2.4.63 **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.
-------------	----------------------------------

12.2.4.64 **void EDMA_HAL_STCDSetChannelMinorLink (edma_software_tcd_t * *stcd*, uint32_t *linkChannel*, bool *enable*)**

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.

12.2.4.65 **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

12.2.4.66 **void EDMA_HAL_PushSTCDToHTCD (DMA_Type * *base*, uint32_t *channel*, edma_software_tcd_t * *stcd*)**

Parameters

<i>base</i>	Register base address for eDMA module.
<i>channel</i>	eDMA channel number.
<i>stcd</i>	The pointer to the software TCD.

12.2.4.67 edma_status_t EDMA_HAL_STCDSetBasicTransfer (DMA_Type * *base*, 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>base</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

12.3 eDMA Peripheral driver

12.3.1 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.

12.3.2 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.

12.3.3 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.

12.3.4 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.

12.3.5 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 TCDs 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_PrepareDescriptorTransfer\(\)](#) 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.

12.3.6 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("Fali 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.

12.3.7 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_edmaBase[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_dmamuxBase[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

- DMA_Type *const [g_edmaBase](#) []
Array for the eDMA module register base address.
- DMAMUX_Type *const [g_dmamuxBase](#) []
Array for DMAMUX module register base address.
- const IRQn_Type [g_edmaIrqId](#) [DMA_INSTANCE_COUNT][FSL_FEATURE_EDMA_MODULE_CHANNEL]
Two dimensional array for eDMA channel interrupt vector number.
- const IRQn_Type [g_edmaErrIrqId](#) [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](#) *user-Config)
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_GetUnfinishedBytes](#) ([edma_chn_state_t](#) *chn)
Gets the unfinished bytes for the eDMA channel current TCD.
- static uint32_t [EDMA_DRV_GetFinishedBytes](#) ([edma_chn_state_t](#) *chn)
Gets the bytes already transferred for the eDMA channel current TCD.

12.3.8 Data Structure Documentation

12.3.8.1 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.
- bool [notHaltOnError](#)
Any error causes the HALT bit to set.

12.3.8.1.0.19 Field Documentation

12.3.8.1.0.19.1 bool [edma_user_config_t::notHaltOnError](#)

Subsequently, all service requests are ignored until the HALT bit is cleared.

12.3.8.2 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.

eDMA Peripheral driver

12.3.8.2.0.20 Field Documentation

12.3.8.2.0.20.1 `uint8_t edma_chn_state_t::channel`

12.3.8.2.0.20.2 `edma_callback_t edma_chn_state_t::callback`

It will be called at the eDMA channel complete and eDMA channel error.

12.3.8.2.0.20.3 `void* edma_chn_state_t::parameter`

12.3.8.2.0.20.4 `volatile edma_chn_status_t edma_chn_state_t::status`

12.3.8.3 struct edma_scatter_gather_list_t

Data Fields

- `uint32_t address`
Address of buffer.
- `uint32_t length`
Length of buffer.

12.3.8.3.0.21 Field Documentation

12.3.8.3.0.21.1 `uint32_t edma_scatter_gather_list_t::address`

12.3.8.3.0.21.2 `uint32_t edma_scatter_gather_list_t::length`

12.3.8.4 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.

12.3.8.4.0.22 Field Documentation

12.3.8.4.0.22.1 `edma_chn_state_t* volatile edma_state_t::chn[FSL_FEATURE_EDMA_DMAMUX_CHANNELS]`

12.3.9 Macro Definition Documentation

12.3.9.1 `#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.

12.3.9.2 `#define VIRTUAL_CHN_TO_EDMA_MODULE_REGBASE(chn) g_edmaBase[chn/FSL_FEATURE_EDMA_MODULE_CHANNEL]`

12.3.9.3 `#define VIRTUAL_CHN_TO_EDMA_CHN(chn) (chn%FSL_FEATURE_EDMA_MODULE_CHANNEL)`

12.3.9.4 `#define VIRTUAL_CHN_TO_DMAMUX_MODULE_REGBASE(chn) g_dmamuxBase[chn/FSL_FEATURE_DMAMUX_MODULE_CHANNEL]`

12.3.9.5 `#define VIRTUAL_CHN_TO_DMAMUX_CHN(chn) (chn%FSL_FEATURE_DMAMUX_MODULE_CHANNEL)`

12.3.10 Typedef Documentation

12.3.10.1 `typedef void(* edma_callback_t)(void *parameter, edma_chn_status_t status)`

Prototype for the callback function registered in the eDMA driver.

12.3.11 Enumeration Type Documentation

12.3.11.1 `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.

12.3.11.2 `enum edma_chn_state_type_t`

Enumerator

- kEDMAInvalidChannel* Macros indicate the failure of the channel request.
- kEDMAAnyChannel* Macros used when requesting channel dynamically.

12.3.11.3 `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.

12.3.12 Function Documentation

12.3.12.1 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 on 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

<i>edmaState</i>	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> .
<i>userConfig</i>	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`.

12.3.12.2 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`.

12.3.12.3 `uint8_t EDMA_DRV_RequestChannel (uint8_t channel, dma_request_source_t source, edma_chn_state_t * chn)`

This function allocates the 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 <code>kEDMAAnyChannel</code> , 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.

12.3.12.4 `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`.

12.3.12.5 `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) [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_STCDSetsMinorLoopOffset\(\)](#) 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>disableDma-Request</i>	Disables (true) or Enable (false) DMA request on TCD complete.

Returns

An eDMA error codes or kStatus_EDMA_Success.

12.3.12.6 static edma_status_t EDMA_DRV_PrepareDescriptorScatterGather (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.

Returns

An eDMA error codes or kStatus_EDMA_Success.

12.3.12.7 static edma_status_t EDMA_DRV_PrepareDescriptorChannelLink (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.

eDMA Peripheral driver

Parameters

<i>stcd</i>	The pointer to the software TCD. The address need to be aligned to 32 bytes.
<i>linkChn</i>	Channel number for major link

Returns

An eDMA error codes or kStatus_EDMA_Success.

12.3.12.8 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.

Returns

An eDMA error codes or kStatus_EDMA_Success.

12.3.12.9 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 and underflow next time it loops to this descriptor. This is an example that describes how to use this interface in audio playback case:

1. Use a ping-pong buffer to transfer the data, the size of the each buffer is 1024 bytes.
2. Each DMA request needs to transfer 8 bytes.
3. The audio data is 16 bit.

```
edma_chn_state_t chn; <--- Simply allocates the structure.
edma_software_tcd_t stcd[2]; <-- Need 32 bytes aligned, two buffer block, needs 2 TCD.
uint32_t srcAddr = buffer; <----Start address of the buffer.
```



```
uint32_t destAddr = SAI_TDR; <-----Destination address, usually SAI TDR register.  
EDMA_DRV_ConfigLoopTransfer(&chn, stcd, kEDMAMemoryToPeripheral, srcAddr, destAddr,  
kEDMATransferSize_2Bytes, 8, 2048, 2) ;
```

eDMA Peripheral driver

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 source memory address.
<i>destAddr</i>	A destination register address or a destination memory address.
<i>size</i>	Bytes to be transferred on every DMA write/read. Source/Dest share the same write/read size.
<i>bytesOnEachRequest</i>	Bytes to be transferred in each DMA request.
<i>totalLength</i>	Total bytes to be transferred in a loop cycle. In audio case, it means the total buffer size.
<i>number</i>	The number of the TCDs, usually in audio case, it means the buffer block number.

Returns

An error code of kStatus_EDMA_Success

12.3.12.10 `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 single-ended 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(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>size</i>	Bytes to be transferred on each DMA write/read. Source/Dest share the same write/read size.
<i>bytesOnEach-Request</i>	Bytes to be transferred in each 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>	The number of TCD memory blocks contained in the scatter gather list.

Returns

An error code of kStatus_EDMA_Success

12.3.12.11 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.

12.3.12.12 edma_status_t EDMA_DRV_StopChannel (edma_chn_state_t * chn)

This function disables the eDMA channel DMA request.

eDMA Peripheral driver

Parameters

<i>chn</i>	The pointer to the channel state structure.
------------	---

Returns

An eDMA error codes or kStatus_EDMA_Success.

12.3.12.13 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.

12.3.12.14 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.
----------------	-------------------------

12.3.12.15 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.

eDMA Peripheral driver

Parameters

<i>instance</i>	eDMA module indicator.
-----------------	------------------------

12.3.12.16 `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.

12.3.12.17 `static uint32_t EDMA_DRV_GetUnfinishedBytes (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 have not finished. 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.

12.3.12.18 `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.

12.3.13 Variable Documentation

12.3.13.1 **DMA_Type* const g_edmaBase[]**

12.3.13.2 **DMAMUX_Type* const g_dmamuxBase[]**

12.3.13.3 **const IRQn_Type g_edmaIrqld[DMA_INSTANCE_COUNT][FSL_FEATURE_EDMA_MODULE_CHANNEL]**

12.3.13.4 **const IRQn_Type g_edmaErrIrqld[DMA_INSTANCE_COUNT]**

eDMA request

12.4 eDMA request

12.4.1 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.

12.4.2 Enumeration Type Documentation

12.4.2.1 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.

12.4.2.2 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 13

Quadrature Encoder/Decoder (ENC)

13.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Quadrature Encoder/Decoder (ENC) block of Kinetis devices.

Modules

- [ENC HAL driver](#)
- [ENC Peripheral Driver](#)

ENC HAL driver

13.2 ENC HAL driver

13.2.1 Overview

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

Enumerations

- enum `enc_status_t` {
`kStatus_ENC_Success` = 0U,
`kStatus_ENC_Error` = 1U,
`kStatus_ENC_InvalidArgument` = 2U }

Encoder status.

- enum `enc_operation_mode_t` {
`kEncNormalMode` = 0U,
`kEncModuloCountingMode` = 1U,
`kEncSignalPhaseCountMode` = 2U }

Encoder operation modes.

- enum `enc_status_flag_t` {
`kEncCmpFlag` = 0U,
`kEncHomeSignalFlag` = 1U,
`kEncWatchdogTimeoutFlag` = 2U,
`kEncIndexPulseFlag` = 3U,
`kEncRollunderFlag` = 4U,
`kEncRolloverFlag` = 5U,
`kEncSimultaneousFlag` = 6U,
`kEncCountDirectionFlag` = 7U }

Encoder status flags.

- enum `enc_int_source_t` {
`kEncIntCmp` = 0U,
`kEncIntHomeSignal` = 1U,
`kEncIntWatchdogTimeout` = 2U,
`kEncIntIndexPulse` = 3U,
`kEncIntRollunder` = 4U,
`kEncIntRollover` = 5U,
`kEncIntSimultaneous` = 6U }

Encoder interrupts.

Configuration

- void `ENC_HAL_Init` (ENC_Type *base)
Resets all configurable registers to be in the reset state for ENC.
- static void `ENC_HAL_SetCmpIntCmd` (ENC_Type *base, bool enable)
Switches to enable the Compare interrupt.
- static bool `ENC_HAL_GetCmpIntCmd` (ENC_Type *base)

- Gets the Compare Interrupt configuration setting.*

 - static bool [ENC_HAL_GetCmpIntFlag](#) (ENC_Type *base)
- Gets the Compare Interrupt Request configuration setting.*

 - static void [ENC_HAL_ClearCmpIntFlag](#) (ENC_Type *base)
- Clears the Compare Interrupt Request bit pending.*

 - static void [ENC_HAL_SetWatchdogCmd](#) (ENC_Type *base, bool enable)
- Switches to enable the Watchdog.*

 - static bool [ENC_HAL_GetWatchdogCmd](#) (ENC_Type *base)
- Gets the Watchdog configuration setting.*

 - static void [ENC_HAL_SetWatchdogIntCmd](#) (ENC_Type *base, bool enable)
- Switches to enable the Watchdog Timeout Interrupt.*

 - static bool [ENC_HAL_GetWatchdogIntCmd](#) (ENC_Type *base)
- Gets the Watchdog Timeout Interrupt configuration setting.*

 - static bool [ENC_HAL_GetWatchdogIntFlag](#) (ENC_Type *base)
- Gets the Watchdog Timeout Interrupt Request configuration setting.*

 - static void [ENC_HAL_ClearWatchdogIntFlag](#) (ENC_Type *base)
- Clears the Watchdog Timeout Interrupt Request pending.*

 - static void [ENC_HAL_SetIndexPulseNegativeEdgeCmd](#) (ENC_Type *base, bool enable)
- Sets the type of INDEX pulse edge.*

 - static bool [ENC_HAL_GetIndexPulseNegativeEdgeCmd](#) (ENC_Type *base)
- Gets INDEX pulse edge configuration setting.*

 - static void [ENC_HAL_SetIndexInitPosCmd](#) (ENC_Type *base, bool enable)
- Switches to enable the INDEX to Initialize Position Counters UPOS and LPOS.*

 - static bool [ENC_HAL_GetIndexInitPosCmd](#) (ENC_Type *base)
- Gets the INDEX to Initialize Position Counters configuration setting.*

 - static void [ENC_HAL_SetIndexPulseIntCmd](#) (ENC_Type *base, bool enable)
- Switches to enable the INDEX Pulse Interrupt.*

 - static bool [ENC_HAL_GetIndexPulseIntCmd](#) (ENC_Type *base)
- Gets the INDEX Pulse Interrupt configuration setting.*

 - static bool [ENC_HAL_GetIndexPulseIntFlag](#) (ENC_Type *base)
- Gets the INDEX Pulse Interrupt Request configuration setting.*

 - static void [ENC_HAL_ClearIndexPulseIntFlag](#) (ENC_Type *base)
- Clears the INDEX Pulse Interrupt Request pending.*

 - static void [ENC_HAL_SetSignalPhaseCountModeCmd](#) (ENC_Type *base, bool enable)
- Enables Signal Phase Count Mode.*

 - static bool [ENC_HAL_GetSignalPhaseCountModeCmd](#) (ENC_Type *base)
- Gets the Signal Phase Count Mode configuration setting.*

 - static void [ENC_HAL_SetReverseCountingCmd](#) (ENC_Type *base, bool enable)
- Switches to enable the Reverse Direction Counting.*

 - static bool [ENC_HAL_GetReverseCountingCmd](#) (ENC_Type *base)
- Gets Direction Counting configuration setting.*

 - static bool [ENC_HAL_GetLastCountDirectionFlag](#) (ENC_Type *base)
- Gets the Last Count Direction Flag.*

 - static void [ENC_HAL_InitPosCounter](#) (ENC_Type *base)
- Initializes the Position Counter.*

 - static void [ENC_HAL_SetHomeSignalNegativeEdgeCmd](#) (ENC_Type *base, bool enable)
- Sets the type of HOME Input Signal Edge.*

 - static bool [ENC_HAL_GetHomeSignalNegativeEdgeCmd](#) (ENC_Type *base)
- Gets HOME Input Signal Edge configuration setting.*

 - static void [ENC_HAL_SetHomeInitPosCmd](#) (ENC_Type *base, bool enable)
- Switches to enable the Initialize Position Counters UPOS and LPOS.*

ENC HAL driver

- static bool [ENC_HAL_GetHomeInitPosCmd](#) (ENC_Type *base)
Gets the HOME to Initialize Position Counters configuration setting.
- static void [ENC_HAL_SetHomeSignalIntCmd](#) (ENC_Type *base, bool enable)
Switches to enable the HOME Signal Interrupt.
- static bool [ENC_HAL_GetHomeSignalIntCmd](#) (ENC_Type *base)
Gets the HOME Signal Interrupt configuration setting.
- static bool [ENC_HAL_GetHomeSignalIntFlag](#) (ENC_Type *base)
Gets the HOME Signal Interrupt Request configuration setting.
- static void [ENC_HAL_ClearHomeSignalIntFlag](#) (ENC_Type *base)
Clears the HOME Signal Interrupt Request pending.
- static void [ENC_HAL_SetInputFilterSampleCount](#) (ENC_Type *base, uint8_t sampleCount)
Sets the Input Filter Sample Count.
- static uint8_t [ENC_HAL_GetInputFilterSampleCount](#) (ENC_Type *base)
Gets the Input Filter Sample Count.
- void [ENC_HAL_SetInputFilterSamplePeriod](#) (ENC_Type *base, uint8_t samplePeriod)
Sets the Input Filter Sample Period.
- static uint8_t [ENC_HAL_GetInputFilterSamplePeriod](#) (ENC_Type *base)
Gets the Input Filter Sample Period.
- static void [ENC_HAL_SetWatchdogTimeout](#) (ENC_Type *base, uint16_t wdtTimeout)
Sets the Watchdog timeout register.
- static uint16_t [ENC_HAL_GetWatchdogTimeout](#) (ENC_Type *base)
Gets the Watchdog timeout register.
- static void [ENC_HAL_SetPosDiffCounterReg](#) (ENC_Type *base, uint16_t diffPosition)
Sets the Position Difference Counter Register.
- static uint16_t [ENC_HAL_GetPosDiffCounterReg](#) (ENC_Type *base)
Gets the Position Difference Counter Register.
- static uint16_t [ENC_HAL_GetPosDiffHoldReg](#) (ENC_Type *base)
Gets the Position Difference Hold Register.
- static void [ENC_HAL_SetRevolutionCounterReg](#) (ENC_Type *base, uint16_t revValue)
Sets the Revolution Counter Register.
- static uint16_t [ENC_HAL_GetRevolutionCounterReg](#) (ENC_Type *base)
Gets the Revolution Counter Register.
- static uint16_t [ENC_HAL_GetRevolutionHoldReg](#) (ENC_Type *base)
Gets the Revolution Hold Register.
- uint32_t [ENC_HAL_GetPosCounterReg](#) (ENC_Type *base)
Gets the Position Counter Register.
- void [ENC_HAL_SetPosCounterReg](#) (ENC_Type *base, uint32_t posVal)
Sets the Position Counter Register.
- uint32_t [ENC_HAL_GetPosHoldReg](#) (ENC_Type *base)
Gets the Position Hold Register.
- void [ENC_HAL_SetInitReg](#) (ENC_Type *base, uint32_t initValue)
Sets the Initialization Register.
- uint32_t [ENC_HAL_GetInitReg](#) (ENC_Type *base)
Gets the Initialization Register.
- static bool [ENC_HAL_GetRawHomeInput](#) (ENC_Type *base)
Gets the Raw HOME Input.
- static bool [ENC_HAL_GetRawIndexInput](#) (ENC_Type *base)
Gets the Raw INDEX Input.
- static bool [ENC_HAL_GetRawPhaseBInput](#) (ENC_Type *base)
Gets the Raw PHASEB Input.
- static bool [ENC_HAL_GetRawPhaseAInput](#) (ENC_Type *base)

- Gets the Raw PHASEA Input.*

 - static bool [ENC_HAL_GetFilteredHomeInput](#) (ENC_Type *base)
- Gets the Filtered HOME Input.*

 - static bool [ENC_HAL_GetFilteredIndexInput](#) (ENC_Type *base)
- Gets the Filtered INDEX Input.*

 - static bool [ENC_HAL_GetFilteredPhaseBInput](#) (ENC_Type *base)
- Gets the Filtered PHASEB Input.*

 - static bool [ENC_HAL_GetFilteredPhaseAInput](#) (ENC_Type *base)
- Gets the Filtered PHASEA Input.*

 - static uint8_t [ENC_HAL_GetTestCount](#) (ENC_Type *base)
- Gets the ENC Test Count.*

 - static void [ENC_HAL_SetTestCount](#) (ENC_Type *base, uint8_t testCount)
- Sets the ENC Test Count.*

 - static uint8_t [ENC_HAL_GetTestPeriod](#) (ENC_Type *base)
- Gets the ENC Test Period.*

 - static void [ENC_HAL_SetTestPeriod](#) (ENC_Type *base, uint8_t testPeriod)
- Sets the ENC Test Period.*

 - static void [ENC_HAL_SetNegativeTestSignalCmd](#) (ENC_Type *base, bool enable)
- Sets the Quadrature Decoder Test Signal.*

 - static bool [ENC_HAL_GetNegativeTestSignalCmd](#) (ENC_Type *base)
- Gets the Quadrature Decoder Test Signal configuration setting.*

 - static void [ENC_HAL_SetTestCounterCmd](#) (ENC_Type *base, bool enable)
- Switches to enable the Test Counter.*

 - static bool [ENC_HAL_GetTestCounterCmd](#) (ENC_Type *base)
- Tests the Test Counter Enable bit.*

 - static void [ENC_HAL_SetTestModuleCmd](#) (ENC_Type *base, bool enable)
- Switches to enable the Test Module.*

 - static bool [ENC_HAL_GetTestModuleCmd](#) (ENC_Type *base)
- Tests the Test Module Enable bit.*

 - void [ENC_HAL_SetModulusReg](#) (ENC_Type *base, uint32_t modValue)
- Sets the ENC Modulus Register.*

 - uint32_t [ENC_HAL_GetModulusReg](#) (ENC_Type *base)
- Gets the ENC Modulus Register.*

 - void [ENC_HAL_SetCmpReg](#) (ENC_Type *base, uint32_t cmpValue)
- Sets the ENC Compare Register.*

 - uint32_t [ENC_HAL_GetCmpReg](#) (ENC_Type *base)
- Gets the ENC Compare Register.*

 - static void [ENC_HAL_SetTriggerUpdateHoldRegCmd](#) (ENC_Type *base, bool enable)
- Switches to enable the Update Hold Registers.*

 - static bool [ENC_HAL_GetTriggerUpdateHoldRegCmd](#) (ENC_Type *base)
- Gets the Update Hold Registers configuration setting.*

 - static void [ENC_HAL_SetTriggerClearPosRegCmd](#) (ENC_Type *base, bool enable)
- Enables Update Position Registers.*

 - static bool [ENC_HAL_GetTriggerClearPosRegCmd](#) (ENC_Type *base)
- Gets the Update Position Registers configuration setting.*

 - static void [ENC_HAL_SetModuloCountingCmd](#) (ENC_Type *base, bool enable)
- Switches to enable the Modulo Counting.*

 - static bool [ENC_HAL_GetModuloCountingCmd](#) (ENC_Type *base)
- Gets the Modulo Counting configuration setting.*

 - static void [ENC_HAL_SetRollunderIntCmd](#) (ENC_Type *base, bool enable)
- Switches to enable the Roll-under Interrupt.*

ENC HAL driver

- static bool [ENC_HAL_GetRollunderIntCmd](#) (ENC_Type *base)
Gets the Roll-under Interrupt configuration setting.
- static bool [ENC_HAL_GetRollunderIntFlag](#) (ENC_Type *base)
Gets the Roll-under Interrupt Request configuration setting.
- static void [ENC_HAL_ClearRollunderIntFlag](#) (ENC_Type *base)
Clears the Roll-under Interrupt Request pending.
- static void [ENC_HAL_SetRolloverIntCmd](#) (ENC_Type *base, bool enable)
Switches to enable the Roll-over Interrupt.
- static bool [ENC_HAL_GetRolloverIntCmd](#) (ENC_Type *base)
Gets the Roll-over Interrupt configuration setting.
- static bool [ENC_HAL_GetRolloverIntFlag](#) (ENC_Type *base)
Gets the Roll-over Interrupt Request configuration setting.
- static void [ENC_HAL_ClearRolloverIntFlag](#) (ENC_Type *base)
Clears the Roll-over Interrupt Request pending.
- static void [ENC_HAL_SetModulusRevCounterCmd](#) (ENC_Type *base, bool enable)
Switches to enable the Modulus Revolution Counter.
- static bool [ENC_HAL_GetModulusRevCounterCmd](#) (ENC_Type *base)
Gets the Modulus Revolution Counter configuration setting.
- static void [ENC_HAL_SetPosmatchOnReadingCmd](#) (ENC_Type *base, bool enable)
Switches to enable the POSMATCH to pulse on Counters registers reading.
- static bool [ENC_HAL_GetPosmatchOnReadingCmd](#) (ENC_Type *base)
Gets the POSMATCH Output configuration setting.
- static void [ENC_HAL_SetSimultaneousIntCmd](#) (ENC_Type *base, bool enable)
Switches to enable the Simultaneous PHASEA and PHASEB Change Interrupt.
- static bool [ENC_HAL_GetSimultaneousIntCmd](#) (ENC_Type *base)
Gets the SAB Interrupt configuration setting.
- static bool [ENC_HAL_GetSimultaneousIntFlag](#) (ENC_Type *base)
Gets the SAB Interrupt Request configuration setting.
- static void [ENC_HAL_ClearSimultaneousIntFlag](#) (ENC_Type *base)
Clears the SAB Interrupt Request pending.

13.2.2 Enumeration Type Documentation

13.2.2.1 enum enc_status_t

Enumerator

- kStatus_ENC_Success* Encoder success status.
- kStatus_ENC_Error* Encoder error status.
- kStatus_ENC_InvalidArgument* Encoder invalid argument.

13.2.2.2 enum enc_operation_mode_t

Enumerator

- kEncNormalMode* Normal mode (transition signal counting).
- kEncModuloCountingMode* Modulo counting mode.
- kEncSignalPhaseCountMode* Signal phase count mode (pulse counting).

13.2.2.3 enum enc_status_flag_t

Enumerator

- kEncCmpFlag* Encoder Compare status flag.
- kEncHomeSignalFlag* Encoder HOME Signal transition status flag.
- kEncWatchdogTimeoutFlag* Encoder Watchdog timeout status flag.
- kEncIndexPulseFlag* Encoder INDEX Pulse transition status flag.
- kEncRollunderFlag* Encoder Roll-under status flag.
- kEncRolloverFlag* Encoder Roll-over status flag.
- kEncSimultaneousFlag* Encoder Simultaneous PHA and PHB change status flag.
- kEncCountDirectionFlag* Encoder Last count direction status flag.

13.2.2.4 enum enc_int_source_t

Enumerator

- kEncIntCmp* Compare interrupt source.
- kEncIntHomeSignal* HOME signal interrupt source.
- kEncIntWatchdogTimeout* Watchdog timeout interrupt source.
- kEncIntIndexPulse* INDEX pulse interrupt source.
- kEncIntRollunder* Roll-under position counter interrupt source.
- kEncIntRollover* Roll-over position counter interrupt source.
- kEncIntSimultaneous* Simultaneous PHASEA and PHASEB change interrupt source.

13.2.3 Function Documentation

13.2.3.1 void ENC_HAL_Init (ENC_Type * base)

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

Parameters

<i>base</i>	The ENC peripheral base address.
-------------	----------------------------------

13.2.3.2 static void ENC_HAL_SetCmplntCmd (ENC_Type * base, bool enable) [inline], [static]

This function allows the user to enable/disable compare interrupt.

ENC HAL driver

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.3 static bool ENC_HAL_GetCmplIntCmd (ENC_Type * *base*) [inline], [static]

This function allows the user to get the compare interrupt configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of compare interrupt setting.

13.2.3.4 static bool ENC_HAL_GetCmplIntFlag (ENC_Type * *base*) [inline], [static]

This function returns the configuration setting of the compare interrupt request. This bit is set when a match occurs between the counter and the COMP value. It will remain set until cleared by software.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Bit setting of the compare interrupt request bit.

13.2.3.5 static void ENC_HAL_ClearCmplIntFlag (ENC_Type * *base*) [inline], [static]

This function clears the compare interrupt request bit.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

**13.2.3.6 static void ENC_HAL_SetWatchdogCmd (ENC_Type * *base*, bool *enable*)
[inline], [static]**

This function allows the user to enable watchdog timer. Allow operation of the watchdog timer monitoring the PHESEA and PHASEB inputs for motor movement.

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

**13.2.3.7 static bool ENC_HAL_GetWatchdogCmd (ENC_Type * *base*) [inline],
[static]**

This function allows the user to get the watchdog configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of watchdog.

**13.2.3.8 static void ENC_HAL_SetWatchdogIntCmd (ENC_Type * *base*, bool *enable*)
[inline], [static]**

This function allows the user to enable watchdog timeout interrupt.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

ENC HAL driver

<i>enable</i>	Bool parameter to enable/disable.
---------------	-----------------------------------

13.2.3.9 static bool ENC_HAL_GetWatchdogIntCmd (ENC_Type * *base*) [inline], [static]

This function allows the user to get the watchdog timeout interrupt configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of wdt timeout interrupt setting.

13.2.3.10 static bool ENC_HAL_GetWatchdogIntFlag (ENC_Type * *base*) [inline], [static]

This function returns the configuration setting of the watchdog timeout interrupt request. This bit is set when a watchdog timeout interrupt occurs. It will remain set until cleared by software. This bit is also cleared when watchdog is disabled.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Bit setting of the wdt timeout interrupt request bit.

13.2.3.11 static void ENC_HAL_ClearWatchdogIntFlag (ENC_Type * *base*) [inline], [static]

This function clears the watchdog timeout interrupt request bit.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

13.2.3.12 static void ENC_HAL_SetIndexPulseNegativeEdgeCmd (ENC_Type * *base*, bool *enable*) [inline], [static]

This function allows the user to set the type of INDEX pulse edge used to initialize the position counter.

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	The edge type of INDEX pulse input.

13.2.3.13 static bool ENC_HAL_GetIndexPulseNegativeEdgeCmd (ENC_Type * *base*) [inline], [static]

This function allows the user to get the type of INDEX pulse edge.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The INDEX pulse edge configuration setting

13.2.3.14 static void ENC_HAL_SetIndexInitPosCmd (ENC_Type * *base*, bool *enable*) [inline], [static]

This function allows the user to enable INDEX pulse to initialize position counters UPOS and LPOS.

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.15 static bool ENC_HAL_GetIndexInitPosCmd (ENC_Type * *base*) [inline], [static]

This function allows the user to get the INDEX to initialize position counters configuration setting.

ENC HAL driver

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of INDEX init position counters.

13.2.3.16 static void ENC_HAL_SetIndexPulseIntCmd (ENC_Type * *base*, bool *enable*) [inline], [static]

This function allows the user to enable the INDEX pulse interrupt.

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.17 static bool ENC_HAL_GetIndexPulseIntCmd (ENC_Type * *base*) [inline], [static]

This function allows the user to get the INDEX pulse interrupt configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of INDEX pulse interrupt setting.

13.2.3.18 static bool ENC_HAL_GetIndexPulseIntFlag (ENC_Type * *base*) [inline], [static]

This function returns the configuration setting of the INDEX pulse interrupt request. This bit is set when an INDEX interrupt occurs. It will remain set until cleared by software.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Bit setting of the INDEX pulse interrupt request bit.

13.2.3.19 static void ENC_HAL_ClearIndexPulseIntFlag (ENC_Type * *base*) [inline], [static]

This function clears the INDEX pulse interrupt request bit.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

13.2.3.20 static void ENC_HAL_SetSignalPhaseCountModeCmd (ENC_Type * *base*, bool *enable*) [inline], [static]

This function allows the user to enable the signal phase count mode which bypasses the quadrature decoder. A positive transition of the PHASEA input generates a count signal. The PHASEB input and the REV (direction control bit) control the counter direction.

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.21 static bool ENC_HAL_GetSignalPhaseCountModeCmd (ENC_Type * *base*) [inline], [static]

This function allows the user to get the signal phase counter mode configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of signal phase count mode setting.

ENC HAL driver

13.2.3.22 `static void ENC_HAL_SetReverseCountingCmd (ENC_Type * base, bool enable) [inline], [static]`

This function allows the user to enable the reverse direction counting. It reverses the interpretation of the quadrature signal, changing the direction of count.

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

**13.2.3.23 static bool ENC_HAL_GetReverseCountingCmd (ENC_Type * *base*)
[inline], [static]**

This function allows the user to get the counting type configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The count type configuration setting.

**13.2.3.24 static bool ENC_HAL_GetLastCountDirectionFlag (ENC_Type * *base*)
[inline], [static]**

This function allows the user to get the flag that indicates the direction of the last count. Returns true if last count was in the up direction or returns false if last count was in the down direction.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of count direction.

**13.2.3.25 static void ENC_HAL_InitPosCounter (ENC_Type * *base*) [inline],
[static]**

This function allows the user to initialize position counters UPOS and LPOS. It will transfer the UINIT and LINIT contents to UPOS and LPOS.

ENC HAL driver

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

13.2.3.26 **static void ENC_HAL_SetHomeSignalNegativeEdgeCmd (ENC_Type * *base*, bool *enable*) [inline], [static]**

This function allows the user to set the type of HOME input signal edge. Use positive or negative going edge-to-trigger initialization of position counters UPOS and LPOS.

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	The edge type of HOME input signal.

13.2.3.27 **static bool ENC_HAL_GetHomeSignalNegativeEdgeCmd (ENC_Type * *base*) [inline], [static]**

This function allows the user to get the HOME input signal edge configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The edge type of HOME input signal.

13.2.3.28 **static void ENC_HAL_SetHomeInitPosCmd (ENC_Type * *base*, bool *enable*) [inline], [static]**

This function allows the user to enable HOME signal to initialize position counters UPOS and LPOS.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

<i>enable</i>	Bool parameter to enable/disable.
---------------	-----------------------------------

13.2.3.29 static bool ENC_HAL_GetHomeInitPosCmd (ENC_Type * *base*) [inline], [static]

This function allows the user to get the HOME signal input init configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of HOME signal initialization POS counters.

13.2.3.30 static void ENC_HAL_SetHomeSignalIntCmd (ENC_Type * *base*, bool *enable*) [inline], [static]

This function allows the user to enable the HOME signal interrupt.

Parameters

<i>base</i>	The ENC module base address
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.31 static bool ENC_HAL_GetHomeSignalIntCmd (ENC_Type * *base*) [inline], [static]

This function allows the user to get the HOME signal interrupt configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of HOME signal interrupt setting.

ENC HAL driver

13.2.3.32 `static bool ENC_HAL_GetHomeSignalIntFlag (ENC_Type * base) [inline],
[static]`

This function returns the configuration setting of the HOME signal interrupt request. This bit is set when a transition on the HOME signal occurs. It will remain set until it is cleared by software.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Bit setting of the HOME signal interrupt request bit.

**13.2.3.33 static void ENC_HAL_ClearHomeSignalIntFlag (ENC_Type * *base*)
[inline], [static]**

This function clears the HOME signal interrupt request bit.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

**13.2.3.34 static void ENC_HAL_SetInputFilterSampleCount (ENC_Type * *base*, uint8_t
sampleCount) [inline], [static]**

This function allows the user to set the input filter sample counts. The value represents the number of consecutive samples that must agree prior to the input filter accepting an input transition. A value of 0x0 represents 3 samples. A value of 0x7 represents 10 samples. A value of *sampleCount* affects the input latency.

Parameters

<i>base</i>	The ENC module base address.
<i>sampleCount</i>	Value that represents the number of consecutive samples.

**13.2.3.35 static uint8_t ENC_HAL_GetInputFilterSampleCount (ENC_Type * *base*)
[inline], [static]**

This function allows the user to read the input filter sample counts. The value represents the number of consecutive samples that must agree prior to the input filter accepting an input transition.

Parameters

ENC HAL driver

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value that represents the number of consecutive samples.

13.2.3.36 void ENC_HAL_SetInputFilterSamplePeriod (ENC_Type * *base*, uint8_t *samplePeriod*)

This function allows the user to set the input filter sample period. This value represents the sampling period of the decoder input signals. Each input is sampled multiple times at the rate specified by this field. If *samplePeriod* is 0x00 (default), then the input filter is bypassed. Bypassing the digital filter enables the position/position difference counters to operate with count rates up to the IPBus frequency. The value of *samplePeriod* affects the input latency.

Parameters

<i>base</i>	The ENC module base address.
<i>samplePeriod</i>	Value of filter sample period.

13.2.3.37 static uint8_t ENC_HAL_GetInputFilterSamplePeriod (ENC_Type * *base*) [inline], [static]

This function allows the user to read the input filter sample period. This value represents the sampling period of the decoder input signals.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of filter sample period.

13.2.3.38 static void ENC_HAL_SetWatchdogTimeout (ENC_Type * *base*, uint16_t *wdtTimeout*) [inline], [static]

This function allows the user to set the timeout value for Watchdog timer, which is separated from the watchdog timer in the COP module. Timeout value is the number of clock cycles plus one that the watchdog timer counts before timing out and optionally generating an interrupt.

Parameters

<i>base</i>	The ENC module base address.
<i>wdtTimeout</i>	Value of watchdog timeout.

**13.2.3.39 static uint16_t ENC_HAL_GetWatchdogTimeout (ENC_Type * *base*)
[inline], [static]**

This function allows the user to read the timeout value for Watchdog timer, which is separated from the watchdog timer in the COP module. Timeout value is the number of clock cycles plus one that the watchdog timer counts before timing out and optionally generating an interrupt.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of watchdog timeout.

**13.2.3.40 static void ENC_HAL_SetPosDiffCounterReg (ENC_Type * *base*, uint16_t
diffPosition) [inline], [static]**

This function allows the user to write the POSD register. It contains the position change in value occurring between each read of the position register. The value of the position difference counter register can be used to calculate velocity.

Parameters

<i>base</i>	The ENC module base address.
<i>diffPosition</i>	Value of position difference.

**13.2.3.41 static uint16_t ENC_HAL_GetPosDiffCounterReg (ENC_Type * *base*)
[inline], [static]**

This function allows the user to read the POSD register. It contains the position change in value occurring between each read of the position register. The value of the position difference counter register can be used to calculate velocity. The 16-bit position difference counter computes up or down on every count pulse. This counter acts as a differentiator whose count value is proportional to the change in position since the last time the position counter was read. When the position register, the position difference counter, or the revolution counter is read, the position difference counter's contents are copied into the position difference hold register (POSDH) and the position difference counter is cleared.

ENC HAL driver

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of position difference hold register.

13.2.3.42 `static uint16_t ENC_HAL_GetPosDiffHoldReg (ENC_Type * base) [inline], [static]`

This function allows the user to read the POSD Hold register. Hold register contains a snapshot of the value of the position difference register. The value of the position difference hold register can be used to calculate velocity.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of position difference hold register.

13.2.3.43 `static void ENC_HAL_SetRevolutionCounterReg (ENC_Type * base, uint16_t revValue) [inline], [static]`

This function allows the user to write the Revolution counter.

Parameters

<i>base</i>	The ENC module base address.
<i>revValue</i>	Value of revolution.

13.2.3.44 `static uint16_t ENC_HAL_GetRevolutionCounterReg (ENC_Type * base) [inline], [static]`

This function allows the user to read the Revolution counter.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of revolution counter.

**13.2.3.45 static uint16_t ENC_HAL_GetRevolutionHoldReg (ENC_Type * *base*)
[inline], [static]**

This function allows the user to read the Revolution Hold register. Contains a snapshot of the value of the revolution counter register.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of revolution hold register.

13.2.3.46 uint32_t ENC_HAL_GetPosCounterReg (ENC_Type * *base*)

This function allows the user to read the Position counter.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of position counter.

13.2.3.47 void ENC_HAL_SetPosCounterReg (ENC_Type * *base*, uint32_t *posVal*)

This function allows the user to write the Position counter.

ENC HAL driver

Parameters

<i>base</i>	The ENC module base address.
<i>posVal</i>	Value of position counter.

13.2.3.48 uint32_t ENC_HAL_GetPosHoldReg (ENC_Type * *base*)

This function allows the user to read the Position hold register. Contains a snapshot of the position counter register.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of position hold register.

13.2.3.49 void ENC_HAL_SetInitReg (ENC_Type * *base*, uint32_t *initValue*)

This function allows the user to write the initialization register.

Parameters

<i>base</i>	The ENC module base address.
<i>initValue</i>	Value of initialization register.

13.2.3.50 uint32_t ENC_HAL_GetInitReg (ENC_Type * *base*)

This function allows the user to read the initialization register.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of initialization register.


```
13.2.3.51 static bool ENC_HAL_GetRawHomeInput ( ENC_Type * base ) [inline],  
          [static]
```

This function allows the user to read the value of the raw HOME input.

ENC HAL driver

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of the raw HOME input.

13.2.3.52 static bool ENC_HAL_GetRawIndexInput (ENC_Type * *base*) [inline], [static]

This function allows the user to read the value of the raw INDEX input.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of the raw INDEX input.

13.2.3.53 static bool ENC_HAL_GetRawPhaseBInput (ENC_Type * *base*) [inline], [static]

This function allows the user to read the value of the raw PHASEB input.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of the raw PHASEB input.

13.2.3.54 static bool ENC_HAL_GetRawPhaseAInput (ENC_Type * *base*) [inline], [static]

This function allows the user to read the value of the raw PHASEA input.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of the raw PHASEA input.

13.2.3.55 static bool ENC_HAL_GetFilteredHomeInput (ENC_Type * *base*) [inline], [static]

This function allows the user to read the value of the filtered HOME input.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of the filtered HOME input.

13.2.3.56 static bool ENC_HAL_GetFilteredIndexInput (ENC_Type * *base*) [inline], [static]

This function allows the user to read the value of the filtered INDEX input.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of the filtered INDEX input.

13.2.3.57 static bool ENC_HAL_GetFilteredPhaseBInput (ENC_Type * *base*) [inline], [static]

This function allows the user to read the value of the filtered PHASEB input.

ENC HAL driver

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of the filtered PHASEB input.

**13.2.3.58 static bool ENC_HAL_GetFilteredPhaseAInput (ENC_Type * *base*)
[inline], [static]**

This function allows the user to read the value of the filtered PHASEA input.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of the filtered PHASEA input.

**13.2.3.59 static uint8_t ENC_HAL_GetTestCount (ENC_Type * *base*) [inline],
[static]**

This function allows the user to read the test count value of the test register. This value holds the number of quadrature advances to generate.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of test count.

**13.2.3.60 static void ENC_HAL_SetTestCount (ENC_Type * *base*, uint8_t *testCount*)
[inline], [static]**

This function allows the user to write the test count value of the test register. This value holds the number of quadrature advances to generate.

Parameters

<i>base</i>	The ENC module base address.
<i>testCount</i>	Value of test count.

13.2.3.61 static uint8_t ENC_HAL_GetTestPeriod (ENC_Type * *base*) [inline], [static]

This function allows the user to read the test period value of the test register. This value holds the period of quadrature phase in IPBus clock cycles.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of test period.

13.2.3.62 static void ENC_HAL_SetTestPeriod (ENC_Type * *base*, uint8_t *testPeriod*) [inline], [static]

This function allows the user to write the test period value of the test register. This value holds the period of quadrature phase in IPBus clock cycles.

Parameters

<i>base</i>	The ENC module base address.
<i>testPeriod</i>	Value of test period.

13.2.3.63 static void ENC_HAL_SetNegativeTestSignalCmd (ENC_Type * *base*, bool *enable*) [inline], [static]

This function allows the user to set the quadrature decoder test signal. Test module can generate quadrature decoder signal in a positive or negative direction.

ENC HAL driver

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	The type of test signal.

13.2.3.64 **static bool ENC_HAL_GetNegativeTestSignalCmd (ENC_Type * *base*) [inline], [static]**

This function allows the user to get the test signal configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The type of test signal.

13.2.3.65 **static void ENC_HAL_SetTestCounterCmd (ENC_Type * *base*, bool *enable*) [inline], [static]**

This function allows the user to enable test counter. It connects the test counter to inputs of the quadrature decoder module.

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.66 **static bool ENC_HAL_GetTestCounterCmd (ENC_Type * *base*) [inline], [static]**

This function returns the configuration setting of the test counter enable bit.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Bit setting of the test counter enable.

13.2.3.67 static void ENC_HAL_SetTestModuleCmd (ENC_Type * *base*, bool *enable*) [inline], [static]

This function allows the user to enable test module. Connects the test module to inputs of the quadrature decoder module.

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.68 static bool ENC_HAL_GetTestModuleCmd (ENC_Type * *base*) [inline], [static]

This function returns the configuration setting of the test module enable bit.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Bit setting of the test module enable.

13.2.3.69 void ENC_HAL_SetModulusReg (ENC_Type * *base*, uint32_t *modValue*)

This function allows the user to write the ENC modulus register. Modulus acts as the upper bound during modulo counting and as the upper reload value when rolling over from the lower bound.

Parameters

<i>base</i>	The ENC module base address.
<i>modValue</i>	Value of modulo register.

ENC HAL driver**13.2.3.70 uint32_t ENC_HAL_GetModulusReg (ENC_Type * *base*)**

This function allows the user to read the ENC modulus register. Modulus acts as the upper bound during modulo counting and as the upper reload value when rolling over from the lower bound.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of modulo register.

13.2.3.71 void ENC_HAL_SetCmpReg (ENC_Type * *base*, uint32_t *cmpValue*)

This function allows the user to write the ENC compare register. When the value of Position counter matches the value of Compare register the CTRL[CMPIRQ] flag is set and the POSMATCH output is asserted.

Parameters

<i>base</i>	The ENC module base address.
<i>cmpValue</i>	Value of modulo register.

13.2.3.72 uint32_t ENC_HAL_GetCmpReg (ENC_Type * *base*)

This function allows the user to read the ENC compare register. When the value of Position counter matches the value of Compare register the CTRL[CMPIRQ] flag is set and the POSMATCH output is asserted.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Value of modulo register.

13.2.3.73 static void ENC_HAL_SetTriggerUpdateHoldRegCmd (ENC_Type * *base*, bool *enable*) [inline], [static]

This function allows the user to enable the update hold registers on external trigger input. Updating POS-DH register will also cause the POSD register to be cleared.

ENC HAL driver

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.74 `static bool ENC_HAL_GetTriggerUpdateHoldRegCmd (ENC_Type * base) [inline], [static]`

This function allows the user to get the update hold registers configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of update hold registers

13.2.3.75 `static void ENC_HAL_SetTriggerClearPosRegCmd (ENC_Type * base, bool enable) [inline], [static]`

This function allows the user to enable the update of position registers on external trigger input. Allows the TRIGGER input to clear POSD, REV, UPOS and LPOS registers on rising edge.

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.76 `static bool ENC_HAL_GetTriggerClearPosRegCmd (ENC_Type * base) [inline], [static]`

This function allows the user to get the update of position registers configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of update position registers

```
13.2.3.77 static void ENC_HAL_SetModuloCountingCmd ( ENC_Type * base, bool enable
) [inline], [static]
```

This function allows the user to enable the modulo counting. It allows the position counters to count in a modulo fashion using MOD and INIT as the upper and lower bounds of the counting range.

ENC HAL driver

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.78 **static bool ENC_HAL_GetModuloCountingCmd (ENC_Type * *base*) [inline], [static]**

This function allows the user to get the modulo counting configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of modulo counting.

13.2.3.79 **static void ENC_HAL_SetRollunderIntCmd (ENC_Type * *base*, bool *enable*) [inline], [static]**

This function allows the user to enable the roll-under interrupt.

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.80 **static bool ENC_HAL_GetRollunderIntCmd (ENC_Type * *base*) [inline], [static]**

This function allows the user to get the roll-under interrupt configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of roll-under interrupt setting.

13.2.3.81 static bool ENC_HAL_GetRollunderIntFlag (ENC_Type * *base*) [inline], [static]

This function returns the configuration setting of the Roll-under interrupt request. It is set when the position counter rolls under from the INIT value to the MOD value or from 0x00000000 to 0xFFFFFFFF. It will remain set until cleared by software.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Bit setting of the interrupt request bit.

13.2.3.82 static void ENC_HAL_ClearRollunderIntFlag (ENC_Type * *base*) [inline], [static]

This function clears the roll-under interrupt request bit.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

13.2.3.83 static void ENC_HAL_SetRolloverIntCmd (ENC_Type * *base*, bool *enable*) [inline], [static]

This function allows the user to enable the roll-over interrupt.

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.84 static bool ENC_HAL_GetRolloverIntCmd (ENC_Type * *base*) [inline], [static]

This function allows the user to get the roll-over interrupt configuration setting.

ENC HAL driver

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of roll-over interrupt setting.

13.2.3.85 `static bool ENC_HAL_GetRolloverIntFlag (ENC_Type * base) [inline], [static]`

This function returns the configuration setting of the Roll-over interrupt request. It is set when the position counter rolls over the MOD value to the INIT value or from 0xFFFFFFFF to 0x00000000. It will remain set until cleared by software.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Bit setting of the interrupt request bit.

13.2.3.86 `static void ENC_HAL_ClearRolloverIntFlag (ENC_Type * base) [inline], [static]`

This function clears the roll-over interrupt request bit.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

13.2.3.87 `static void ENC_HAL_SetModulusRevCounterCmd (ENC_Type * base, bool enable) [inline], [static]`

This function allows the user to enable the modulo revolution counter. This is used to determine how the revolution counter (REV) is incremented or decremented. By default REV is controlled based on the count direction and the INDEX pulse. As an option, REV can be controlled using the roll-over/under detection during modulo counting.

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.88 **static bool ENC_HAL_GetModulusRevCounterCmd (ENC_Type * *base*) [inline], [static]**

This function allows the user to get the modulus revolution counter configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of modulus revolution counter.

13.2.3.89 **static void ENC_HAL_SetPosmatchOnReadingCmd (ENC_Type * *base*, bool *enable*) [inline], [static]**

This function allows the user to config control of the POSMATCH output. POSMATCH pulses when the UPOS, LPOS, REV or POSD registers are read - when set true or when match occurred between position register and Compare value register (false).

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.90 **static bool ENC_HAL_GetPosmatchOnReadingCmd (ENC_Type * *base*) [inline], [static]**

This function allows the user to get the POSMATCH output configuration setting.

Parameters

ENC HAL driver

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of POSMATCH output setting.

13.2.3.91 static void ENC_HAL_SetSimultaneousIntCmd (ENC_Type * *base*, bool *enable*) [inline], [static]

This function allows the user to enable the SAB interrupt.

Parameters

<i>base</i>	The ENC module base address.
<i>enable</i>	Bool parameter to enable/disable.

13.2.3.92 static bool ENC_HAL_GetSimultaneousIntCmd (ENC_Type * *base*) [inline], [static]

This function allows the user to get the SAB interrupt configuration setting.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

The state of SAB interrupt setting.

13.2.3.93 static bool ENC_HAL_GetSimultaneousIntFlag (ENC_Type * *base*) [inline], [static]

This function returns the configuration setting of the SAB interrupt request. It indicates that the PHASEA and PHASEB inputs changed simultaneously (within a single clock period). This event typically indicates an error condition because quadrature coding requires only one of these inputs to change at a time. The bit remains set until it is cleared by software or a reset.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

Returns

Bit setting of the interrupt request bit.

**13.2.3.94 static void ENC_HAL_ClearSimultaneousIntFlag (ENC_Type * *base*)
[inline], [static]**

This function clears the SAB interrupt request bit.

Parameters

<i>base</i>	The ENC module base address.
-------------	------------------------------

ENC Peripheral Driver

13.3 ENC Peripheral Driver

13.3.1 Overview

The section describes the programming interface of the ENC Peripheral driver.

13.3.2 Overview

The enhanced Quadrature Encoder/Decoder module provides interfacing capability to position/speed sensors used in industrial motor control applications. It has four input signals: PHASEA, PHASEB, INDEX, and HOME. This module is used to decode shaft position, revolution count, and speed.

13.3.3 Initialization

To initialize the ENC module, call the [ENC_DRV_Init\(\)](#) function and pass the configuration data structure ([enc_user_config_t](#)), which can be filled by the [ENC_DRV_StructInitUserConfigNormal\(\)](#) function with the default settings for the encoder. After it is initialized, the ENC module can function as an ENC decoder. The default settings are as follows.

```

enc_state_t encUserState;
enc_user_config_t encUserConfig;

encUserConfig.operationMode = kEncNormalMode;           /* Operation mode: Normal
mode, modulo counting mode or bypass (signal phase count) mode. */
encUserConfig.reverseCounting = false;                 /* Counting direction: Normal (false) or
reverse (true) counting direction. */
encUserConfig.indexInputNegativeEdge = false;         /* Type of transition edge of
INDEX input signal: positive (false) or negative (true). */
encUserConfig.homeInputNegativeEdge = false;         /* Type of transition edge of HOME
input signal: positive (false) or negative (true). */
encUserConfig.indexPulsePosInit = true;              /* To use HOME (false) or INDEX (true)
input to initialize position counter to value in Initialization Register. */
encUserConfig.posCntInitValue = 0;                   /* Value to put in Initialization
Register. */
encUserConfig.posCmpValue = 0xFFFF;                 /* Value to put in Position Compare
Register. */
encUserConfig.moduloValue = 0;                       /* Value to put in Modulus Register. */
encUserConfig.triggerUpdateHoldRegEnable = false;    /* Enable/disable updating
hold registers on TRIGGER input. */
encUserConfig.triggerClearPosRegEnable = false;     /* Enable/disable clear of
position registers on TRIGGER input. */
encUserConfig.moduloRevolutionCounting = false;     /* Type of revolution counter -
index pulse counting (on false) or modulo counting (on true). */
encUserConfig.outputControlOnReading = false;       /* Used to control the behaviour
of the POSMATCH output signal. True - output control on reading position register, false - OC on match
position register. */
encUserConfig.watchdogTimeout = 0;                  /* Value to put in Watchdog Timeout
Register. */
encUserConfig.filterCount = 0;                      /* Value represents the number of
consecutive samples that must agree prior to the input filter accepting an input transition. */
encUserConfig.filterPeriod = 0;                    /* Value represents the sampling period (in
IPBus clock cycles) of the decoder input signals. */

```

Ensure that any default settings are changed after calling the [ENC_DRV_StructInitUserConfigNormal\(\)](#) function. For example, changing INDEX input edge type and position compare register value:

```
encUserConfig.indexPulsePosInit = true;  
encUserConfig.posCmpValue = 0x1010;
```

After that, the initialization is completed by calling the initialization function:

```
ENC_DRV_Init(instance, &encUserConfig, &encUserState);
```

To de-initialize the ENC peripheral, call the `ENC_DRV_Deinit()` function which shuts down the ENC clock to reduce the power consumption.

13.3.4 Testing ENC module

The ENC peripheral provides a test module to drive the ENC quadrature inputs which can provide a mechanism for customers to test the ENC module is functioning without being connected in the intended "incremental position sensor" for motor control.

To initialize the ENC Test module, call the `ENC_DRV_TestInit()` function and pass the configuration data structure (`enc_test_config_t`), which can be filled as the example shows.

```
enc_test_config_t encTestConfig;  
  
encTestConfig.testNegativeSignalEnable = false;  
encTestConfig.testCount = 100;  
encTestConfig.testPeriod = 10;  
  
ENC_DRV_TestInit(instance, &encTestConfig);
```

Test module generates 100 quadrature advances into quadrature inputs with period 10 IPBus clock cycles.

13.3.5 Input monitor

ENC peripheral also provides monitoring of input signals. It can be used as follows.

```
enc_input_monitor_t encInputMonitor;  
  
ENC_DRV_ReadInputMonitorRegister(instance, true, &encInputMonitor);
```

The `encInputMonitor` structure contains the values of the raw (if true) or filtered (if false) PHASEA, PHASEB, INDEX and HOME input signals and the reset value of the raw and filtered values of PHASEA, PHASEB, INDEX, and HOME. If the input pins are connected to a pull-up, IMR 0-7 bits are set to one (1). If these input pins are connected to a pull-down device, the 0-7 bits are set to zeroes (0).

ENC Peripheral Driver

13.3.6 Interrupts

Interrupt may be enabled/disabled by calling the [ENC_DRV_SetIntMode\(\)](#) function and passing the interrupt source `enc_interrupt_source_t` argument. The user callback function can be installed by calling the [ENC_DRV_InstallCallback\(\)](#) function and passing the name as the type `enc_callback_t` argument. The parameter of user callback function may be passed too.

ENC peripheral provides these type of interrupts:

1. Position compare interrupt
2. HOME signal transition interrupt
3. Watchdog time-out interrupt
4. INDEX pulse transition interrupt
5. Roll-under interrupt
6. Roll-over interrupt
7. Simultaneous PHASEA and PHASEB change interrupt

To get a status of any interrupt source use the [ENC_DRV_GetIntMode\(\)](#) function. The flag status reading is provided by the [ENC_DRV_GetStatusFlag\(\)](#) function and by clearing the flag with the [ENC_DRV_ClearStatusFlag\(\)](#) function.

13.3.7 Reading Counters

To read the counters, call the [ENC_DRV_ReadCounters\(\)](#) function and pass the argument of the `enc_counter_t` type. This function reads the position difference counter register that causes snapshots of the position, position difference counter and revolution counter registers are each placed into their respective hold registers. Then, the position counter value and revolution counter value are read from their hold registers. These three values are stored into a structure and passed as an argument of the function.

To reset counters (position, position difference, and revolution) use the [ENC_DRV_ResetCounters\(\)](#) function.

13.3.8 Reading Hold Registers

To read an action, such as the trigger input signal, which causes getting snapshots of the position counter, position difference counter, and revolution counter register to their respective hold registers, use the [ENC_DRV_ReadHoldRegisters\(\)](#) function.

Data Structures

- struct [enc_user_config_t](#)
User configuration structure for ENC driver. [More...](#)
- struct [enc_test_config_t](#)
User configuration structure for ENC driver - ENC test module configuration. [More...](#)
- struct [enc_counter_t](#)

- *Counter registers structure for ENC driver. [More...](#)*
- struct [enc_input_monitor_t](#)
Input monitor structure for ENC driver. [More...](#)

Configuration

- [enc_status_t ENC_DRV_StructInitUserConfigNormal](#) ([enc_user_config_t](#) *userConfigPtr)
Fills the initial user configuration for the ENC module without the interrupts enablement.
- [enc_status_t ENC_DRV_Init](#) (uint32_t instance, const [enc_user_config_t](#) *userConfigPtr)
Initializes an ENC instance for operation.
- void [ENC_DRV_Deinit](#) (uint32_t instance)
De-initializes the ENC peripheral.
- [enc_status_t ENC_DRV_TestInit](#) (uint32_t instance, const [enc_test_config_t](#) *userConfigPtr)
Initializes an ENC test module.
- void [ENC_DRV_TestDeinit](#) (uint32_t instance)
Shuts down the ENC test module, disables test counter, and clears the test period and test count values.
- [enc_status_t ENC_DRV_SetIntMode](#) (uint32_t instance, [enc_int_source_t](#) intSrc, bool enable)
Enables/disables the selected ENC interrupt.
- bool [ENC_DRV_GetIntMode](#) (uint32_t instance, [enc_int_source_t](#) intSrc)
Gets the configuration of the selected ENC interrupt.
- bool [ENC_DRV_GetStatusFlag](#) (uint32_t instance, [enc_status_flag_t](#) flag)
Gets the interrupt status flag of the selected interrupt source.
- void [ENC_DRV_ClearStatusFlag](#) (uint32_t instance, [enc_status_flag_t](#) flag)
Clears the status flag of the selected status source.
- [enc_status_t ENC_DRV_ReadCounters](#) (uint32_t instance, [enc_counter_t](#) *countRegPtr)
Reads the actual values of the ENC counter registers.
- [enc_status_t ENC_DRV_ReadHoldReg](#) (uint32_t instance, [enc_counter_t](#) *holdRegPtr)
Reads the ENC hold registers.
- void [ENC_DRV_ResetCounters](#) (uint32_t instance)
Resets the ENC counter registers.
- [enc_status_t ENC_DRV_ReadInputMonitor](#) (uint32_t instance, bool inputMonitorRaw, [enc_input_monitor_t](#) *inputMonitorPtr)
Reads the ENC input monitor register.

13.3.9 Data Structure Documentation

13.3.9.1 struct [enc_user_config_t](#)

Use an instance of this structure with the [ENC_DRV_Init\(\)](#) function. This enables configuration of the most common settings of the ENC peripheral with a single function call.

Data Fields

- uint32_t [posCntInitValue](#)
Value to put in Initialization Register.
- uint32_t [posCmpValue](#)
Value to put in Position Compare Register.

ENC Peripheral Driver

- uint32_t [moduloValue](#)
Value to put in Modulus Register.
- uint16_t [watchdogTimeout](#)
Value to put in Watchdog Timeout Register.
- uint8_t [filterCount](#)
Value represents the number of consecutive samples that must agree prior to the input filter accepting an input transition.
- uint8_t [filterPeriod](#)
Value represents the sampling period (in IPBus clock cycles) of the decoder input signals.
- [enc_operation_mode_t](#) [operationMode](#)
Operation mode: Normal mode, modulo counting mode or bypass (signal phase count) mode.
- bool [reverseCounting](#)
Counting direction: Normal (false) or reverse (true) counting direction.
- bool [indexInputNegativeEdge](#)
Type of transition edge of INDEX input signal: positive (false) or negative (true).
- bool [homeInputNegativeEdge](#)
Type of transition edge of HOME input signal: positive (false) or negative (true).
- bool [indexPulsePosInit](#)
To use HOME (false) or INDEX (true) input to initialize position counter to value in Initialization Register.
- bool [triggerUpdateHoldRegEnable](#)
Enable/disable updating hold registers on TRIGGER input.
- bool [triggerClearPosRegEnable](#)
Enable/disable clear of position registers on TRIGGER input.
- bool [moduloRevolutionCounting](#)
Type of revolution counter - index pulse counting (on false) or modulo counting (on true).
- bool [outputControlOnReading](#)
Used to control the behaviour of the POSMATCH output signal.

13.3.9.1.0.23 Field Documentation

13.3.9.1.0.23.1 bool [enc_user_config_t::outputControlOnReading](#)

True - output control on reading position register, false - OC on match position register.

13.3.9.2 struct [enc_test_config_t](#)

Use an instance of this structure with the [ENC_DRV_TestInit\(\)](#) function. This enables configuration of the Test module of the ENC peripheral with a single function call.

Data Fields

- uint8_t [testCount](#)
Test count - holds the number of quadrature advances to generate.
- uint8_t [testPeriod](#)
Test period - holds the period of quadrature phase in IPBus clock cycles.
- bool [testNegativeSignalEnable](#)
Test signal type, positive (false) or negative (true).

13.3.9.2.0.24 Field Documentation**13.3.9.2.0.24.1** `uint8_t enc_test_config_t::testCount`**13.3.9.2.0.24.2** `uint8_t enc_test_config_t::testPeriod`**13.3.9.2.0.24.3** `bool enc_test_config_t::testNegativeSignalEnable`**13.3.9.3 struct enc_counter_t**

Use an instance of this structure with the `ENC_DRV_ReadHoldRegisters()` or `ENC_DRV_ReadCounters()` functions. This reads counters and hold registers of Position, PositionDifference, Revolution Counter.

Data Fields

- `int32_t position`
Position Counter/Hold Register.
- `int16_t posDiff`
Position Difference Counter/Hold Register.
- `int16_t revolution`
Revolution Counter/Hold Register.

13.3.9.3.0.25 Field Documentation**13.3.9.3.0.25.1** `int32_t enc_counter_t::position`**13.3.9.3.0.25.2** `int16_t enc_counter_t::posDiff`**13.3.9.3.0.25.3** `int16_t enc_counter_t::revolution`**13.3.9.4 struct enc_input_monitor_t**

Use an instance of this structure with the `ENC_DRV_ReadInputMonitorRegister()`. This reads Input Monitor register that contains the values of the raw or filtered PHASEA, PHASEB, INDEX and HOME input signals.

Data Fields

- `bool phaseA`
PHASEA input.
- `bool phaseB`
PHASEB input.
- `bool index`
INDEX input.
- `bool home`
HOME input.

ENC Peripheral Driver

13.3.9.4.0.26 Field Documentation

13.3.9.4.0.26.1 `bool enc_input_monitor_t::phaseA`

13.3.9.4.0.26.2 `bool enc_input_monitor_t::phaseB`

13.3.9.4.0.26.3 `bool enc_input_monitor_t::index`

13.3.9.4.0.26.4 `bool enc_input_monitor_t::home`

13.3.10 Function Documentation

13.3.10.1 `enc_status_t ENC_DRV_StructInitUserConfigNormal (enc_user_config_t * userConfigPtr)`

This function fills the initial user configuration. Calling the initialization function with the filled parameter configures the ENC module to function as a simple Quadrature Encoder. The settings are:

```
encUserConfig.operationMode = kEncNormalMode;
encUserConfig.reverseCounting = false;
encUserConfig.indexInputNegativeEdge = false;
encUserConfig.homeInputNegativeEdge = false;
encUserConfig.indexPulsePosInit = true;
encUserConfig.posCntInitValue = 0U;
encUserConfig.posCmpValue = 0xFFFFU;
encUserConfig.moduloValue = 0U;
encUserConfig.triggerUpdateHoldRegEnable = false;
encUserConfig.triggerClearPosRegEnable = false;
encUserConfig.moduloRevolutionCounting = false;
encUserConfig.outputControlOnReading = false;
encUserConfig.watchdogTimeout = 0U;
encUserConfig.filterCount = 0U;
encUserConfig.filterPeriod = 0U;
```

Parameters

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

Returns

Execution status.

13.3.10.2 `enc_status_t ENC_DRV_Init (uint32_t instance, const enc_user_config_t * userConfigPtr)`

This function initializes the run-time state structure to un-gate the clock to the ENC 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. This example shows how to set up the `enc_state_t` and the `enc_user_config_t` parameters and how to call the `ENC_DRV_Init` function by passing in these parameters:


```

enc_user_config_t encUserConfig;
encUserConfig.operationMode = kEncNormalMode;
encUserConfig.reverseCounting = false;
encUserConfig.indexInputNegativeEdge = false;
encUserConfig.homeInputNegativeEdge = false;
encUserConfig.indexPulsePosInit = true;
encUserConfig.posCntInitValue = 0U;
encUserConfig.posCmpValue = 0xFFFFU;
encUserConfig.moduloValue = 0U;
encUserConfig.triggerUpdateHoldRegEnable = false;
encUserConfig.triggerClearPosRegEnable = false;
encUserConfig.moduloRevolutionCounting = false;
encUserConfig.outputControlOnReading = false;
encUserConfig.watchdogTimeout = 0U;
encUserConfig.filterCount = 0U;
encUserConfig.filterPeriod = 0U;
ENC_DRV_Init (&encUserConfig);

```

Parameters

<i>instance</i>	ENC instance ID.
<i>userConfigPtr</i>	The user configuration structure of type <code>enc_user_config_t</code> . The user is responsible to fill out the members of this structure and to pass the pointer of this structure into this function.

Returns

Execution status.

13.3.10.3 void ENC_DRV_Deinit (uint32_t instance)

This function shuts down the ENC clock to reduce power consumption.

Parameters

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

13.3.10.4 enc_status_t ENC_DRV_TestInit (uint32_t instance, const enc_test_config_t * userConfigPtr)

This function initializes the run-time state structure to enable the test module and sets the test period and test count values. This example shows how to set up the `enc_test_config_t` parameters and how to call the `ENC_DRV_TestInit` function by passing in these parameters:

```

enc_test_config_t encTestConfig;
encTestConfig.testNegativeSignalEnable = false;
encTestConfig.testCount = 100;
encTestConfig.testPeriod = 10;
ENC_DRV_TestInit (&encTestConfig);

```

ENC Peripheral Driver

Parameters

<i>instance</i>	ENC instance ID.
<i>userConfigPtr</i>	The user configuration structure of type enc_test_config_t .

Returns

Execution status.

13.3.10.5 void ENC_DRV_TestDeinit (uint32_t instance)

Parameters

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

13.3.10.6 enc_status_t ENC_DRV_SetIntMode (uint32_t instance, enc_int_source_t intSrc, bool enable)

The interrupt source is passing as argument of type [enc_interrupt_source_t](#).

Parameters

<i>instance</i>	ENC instance ID.
<i>intSrc</i>	The type of interrupt to enable/disable.
<i>enable</i>	Bool parameter to enable/disable.

Returns

Execution status.

13.3.10.7 bool ENC_DRV_GetIntMode (uint32_t instance, enc_int_source_t intSrc)

The interrupt type is passing as an argument of type [enc_int_source_t](#).

Parameters

<i>instance</i>	ENC instance ID.
<i>intSrc</i>	The type of interrupt to get configuration.

Returns

Configuration of selected interrupt source.

13.3.10.8 **bool ENC_DRV_GetStatusFlag (uint32_t *instance*, enc_status_flag_t *flag*)**

Parameters

<i>instance</i>	ENC instance ID.
<i>flag</i>	Selected type of status flag.

Returns

State of selected flag.

13.3.10.9 **void ENC_DRV_ClearStatusFlag (uint32_t *instance*, enc_status_flag_t *flag*)**

Parameters

<i>instance</i>	ENC instance ID.
<i>flag</i>	Selected type of status flag.

13.3.10.10 **enc_status_t ENC_DRV_ReadCounters (uint32_t *instance*, enc_counter_t * *countRegPtr*)**

Parameters

<i>instance</i>	ENC instance ID.
<i>countRegPtr</i>	The structure of ENC counter registers.

Returns

Execution status.

13.3.10.11 **enc_status_t ENC_DRV_ReadHoldReg (uint32_t *instance*, enc_counter_t * *holdRegPtr*)**

ENC Peripheral Driver

Parameters

<i>instance</i>	ENC instance ID.
<i>holdRegPtr</i>	The structure of ENC hold registers.

Returns

Execution status.

13.3.10.12 void ENC_DRV_ResetCounters (uint32_t *instance*)

Parameters

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

13.3.10.13 enc_status_t ENC_DRV_ReadInputMonitor (uint32_t *instance*, bool *inputMonitorRaw*, enc_input_monitor_t * *inputMonitorPtr*)

Parameters

<i>instance</i>	ENC instance ID.
<i>inputMonitorRaw</i>	The type of input monitor - raw (true) / filtered (false).
<i>inputMonitorPtr</i>	The structure of ENC Monitor register variables.

Returns

Execution status.

Chapter 14

Ethernet MAC (ENET)

14.1 Overview

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

Modules

- [ENET HAL driver](#)
- [ENET Peripheral Driver](#)
- [ENET Physical Layer Driver](#)
- [ENET RTCS Adaptor](#)

ENET HAL driver

14.2 ENET HAL driver

14.2.1 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 IEEE 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...](#)
- struct [enet_bd_config](#)
The configuration structure of buffer descriptor. [More...](#)
- struct [enet_cur_status_t](#)
The structure to save current status. [More...](#)
- struct [enet_bd_attr_t](#)
The buffer descriptor attribute. [More...](#)

Macros

- #define [ENET_ALIGN](#)(x, align) ((unsigned int)((x) + ((align)-1)) & (unsigned int)(~(unsigned int)((align)- 1)))
Defines the system endian type.
- #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` {


```

kEnetRxBdInterrupt = 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` {


```

kEnetTx BdReady = 0x8000U,
kEnetTx BdTxSoftOwner1 = 0x4000U,
kEnetTx BdWrap = 0x2000U,
kEnetTx BdTxSoftOwner2 = 0x1000U,
kEnetTx BdLast = 0x0800U,
kEnetTx BdTransmitCrc = 0x0400U }

```

Defines the control status of the transmit buffer descriptor.

- enum `enet_tx_bd_control_extend0_t` {


```

kEnetTx BdTxErr = 0x8000U,
kEnetTx BdTxUnderFlowErr = 0x2000U,
kEnetTx BdExcessCollisionErr = 0x1000U,
kEnetTx BdTxFrameErr = 0x0800U,
kEnetTx BdLatecollisionErr = 0x0400U,
kEnetTx BdOverflowErr = 0x0200U,
kEnetTx TimestampErr = 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 IEEE 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 IEEE 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/TSVAAIL interrupt source for ENET.*

- 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 {
 

```

kEnetTxAccelShift16Enabled = 0x01U,
kEnetTxAccelIpCheckEnabled = 0x08U,
kEnetTxAccelProtoCheckEnabled = 0x10U }

```

*Defines the transmit accelerator configuration.*

- enum enet\_rxaccelerator\_config\_t {
 

```

kEnetRxAccelPadRemoveEnabled = 0x01U,
kEnetRxAccelIpCheckEnabled = 0x02U,
kEnetRxAccelProtoCheckEnabled = 0x04U,
kEnetRxAccelMacCheckEnabled = 0x40U,
kEnetRxAccelShift16Enabled = 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.*

  - enum `enet_en_dynamical_act_t` {  
`kEnGraceSendStop`,  
`kEnSendPauseFrame`,  
`kEnClearMibCounter` }

*The action of mac which should be enabled dynamically.*

## Functions

- `enet_status_t` `ENET_HAL_Init` (`ENET_Type *base`)  
*Initializes the ENET module to reset status.*
  - void `ENET_HAL_Config` (`ENET_Type *base`, const `enet_mac_config_t *macCfgPtr`, const `uint32_t sysClk`, const `enet_bd_config *bdConfig`)  
*Configures the ENET.*
  - void `ENET_HAL_GetStatus` (`ENET_Type *base`, const `uint32_t mask`, `enet_cur_status_t *curStatus`)  
*Gets the ENET status.*
  - void `ENET_HAL_SetMulticastAddrHash` (`ENET_Type *base`, `uint32_t crcValue`, `enet_special_address_filter_t mode`)  
*Sets the hardware addressing filtering to a multicast group address.*
  - void `ENET_HAL_GetBufDescripAttr` (volatile `enet_bd_struct_t *curBd`, const `uint64_t mask`, `enet_bd_attr_t *resultAttr`)  
*Gets the attribute field value of buffer descriptor structure and flag status in the control field.*
  - `uint8_t *` `ENET_HAL_GetBufDescripData` (volatile `enet_bd_struct_t *curBd`)  
*Gets the buffer address of the buffer descriptors.*
  - void `ENET_HAL_ClrRxBdAfterHandled` (volatile `enet_bd_struct_t *rxBds`, `uint8_t *data`, bool `isbufferUpdate`)  
*Clears the receive buffer descriptor flag after it has been received or encountered some error in the receiving process.*
  - void `ENET_HAL_SetTxBdBeforesend` (volatile `enet_bd_struct_t *txBds`, `uint16_t length`, bool `isTxTsCfged`, bool `isTxCrcEnable`, bool `isLastOne`)  
*Sets the transmit buffer descriptor flag before sending a frame.*
  - static void `ENET_HAL_ClrTxBdAfterSend` (volatile `enet_bd_struct_t *curBd`)  
*Clears the context in the transmit buffer descriptors.*
  - static void `ENET_HAL_SetRxBdActive` (`ENET_Type *base`)  
*Activates the receive buffer descriptor.*
  - static void `ENET_HAL_SetTxBdActive` (`ENET_Type *base`)  
*Activates the transmit buffer descriptor.*
  - void `ENET_HAL_SetRMII Mode` (`ENET_Type *base`, `enet_config_rmii_t *rmiiCfgPtr`)  
*Configures the (R)MII data interface of ENET.*
  - static `uint32_t` `ENET_HAL_GetSMIData` (`ENET_Type *base`)  
*Reads data from PHY.*
  - void `ENET_HAL_SetSMIRead` (`ENET_Type *base`, `uint32_t phyAddr`, `uint32_t phyReg`, `enet_mii_read_t operation`)  
*Sets the SMI(serial Management interface) read command.*
  - void `ENET_HAL_SetSMIWrite` (`ENET_Type *base`, `uint32_t phyAddr`, `uint32_t phyReg`, `enet_mii_write_t operation`, `uint32_t data`)  
*Sets the SMI(serial Management interface) write command.*

- void [ENET\\_HAL\\_EnDynamicalAct](#) (ENET\_Type \*base, [enet\\_en\\_dynamical\\_act\\_t](#) action, bool enable)  
*Enables/disables the MAC dynamical action.*
- static void [ENET\\_HAL\\_Enable](#) (ENET\_Type \*base)  
*Enables the ENET module.*
- static void [ENET\\_HAL\\_Disable](#) (ENET\_Type \*base)  
*Disables the ENET module.*
- void [ENET\\_HAL\\_SetIntMode](#) (ENET\_Type \*base, [enet\\_interrupt\\_request\\_t](#) source, bool enable)  
*Enables/Disables the ENET interrupt.*
- static void [ENET\\_HAL\\_ClearIntStatusFlag](#) (ENET\_Type \*base, [enet\\_interrupt\\_request\\_t](#) source)  
*Clears ENET interrupt events.*
- static bool [ENET\\_HAL\\_GetIntStatusFlag](#) (ENET\_Type \*base, [enet\\_interrupt\\_request\\_t](#) source)  
*Gets the ENET interrupt status.*
- void [ENET\\_HAL\\_Start1588Timer](#) (ENET\_Type \*base, [enet\\_config\\_ptp\\_timer\\_t](#) \*ptpCfgPtr)  
*Configures the IEEE 1588 timer and run the IEEE 1588 timer.*
- void [ENET\\_HAL\\_Stop1588Timer](#) (ENET\_Type \*base)  
*Stop the IEEE 1588 timer.*
- static void [ENET\\_HAL\\_Adjust1588Timer](#) (ENET\_Type \*base, uint32\_t increaseCorrection, uint32\_t periodCorrection)  
*Adjusts the IEEE 1588 timer.*
- static void [ENET\\_HAL\\_Set1588TimerNewTime](#) (ENET\_Type \*base, uint32\_t nanSecond)  
*Sets the IEEE 1588 timer.*
- static uint32\_t [ENET\\_HAL\\_Get1588TimerCurrentTime](#) (ENET\_Type \*base)  
*Gets the time from the IEEE 1588 timer.*
- static bool [ENET\\_HAL\\_Get1588TimerChnStatus](#) (ENET\_Type \*base, [enet\\_timer\\_channel\\_t](#) channel)  
*Gets the IEEE 1588 timer channel status.*
- static void [ENET\\_HAL\\_Rst1588TimerCmpValAndClrFlag](#) (ENET\_Type \*base, [enet\\_timer\\_channel\\_t](#) channel, uint32\_t compareValue)  
*Resets the IEEE 1588 timer compare value and clears the IEEE 1588 timer channel interrupt flag.*

## 14.2.2 Data Structure Documentation

### 14.2.2.1 struct enet\_bd\_struct\_t

#### Data Fields

- uint16\_t [length](#)  
*Buffer descriptor data length.*
- 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.*

## ENET HAL driver

- uint8\_t **headerLength**  
*Header length.*
- uint8\_t **protocolType**  
*Protocol type.*
- uint16\_t **controlExtend2**  
*Extend buffer descriptor control2.*
- uint32\_t **timestamp**  
*Timestamp.*

### 14.2.2.2 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.*

### 14.2.2.3 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.*

### 14.2.2.4 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.*

#### 14.2.2.4.0.27 Field Documentation

##### 14.2.2.4.0.27.1 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

#### 14.2.2.5 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.*

#### 14.2.2.6 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**  
*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.*

## ENET HAL driver

- [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.*

### 14.2.2.7 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 [ieeetxFrameMultiCollison](#)  
*Transmit frames with multicast collision.*
- uint16\_t [ieeetxFrameLateCollison](#)  
*Transmit frames with late collision.*
- uint16\_t [ieeetxFrmaeExcCollison](#)  
*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.*

#### 14.2.2.8 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.*

## ENET HAL driver

### 14.2.2.9 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.*
- `uint32_t macCtlConfigure`  
*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*
- [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*

### 14.2.2.10 struct enet\_bd\_config

#### Data Fields

- volatile [enet\\_bd\\_struct\\_t](#) \* `txBds`  
*The start address of ENET transmit buffer descriptors.*
- `uint8_t * txBuffer`  
*The transmit data buffer start address.*
- `uint32_t txBdNumber`  
*The transmit buffer descriptor numbers.*
- `uint32_t txBuffSizeAlign`  
*The aligned transmit buffer size.*
- volatile [enet\\_bd\\_struct\\_t](#) \* `rxBds`  
*The start address of ENET receive buffer descriptors.*
- `uint8_t * rxBuffer`  
*The receive data buffer start address.*
- `uint32_t rxBdNumber`  
*The receive buffer descriptor numbers.*
- `uint32_t rxBuffSizeAlign`  
*The aligned receive transmit buffer size.*

#### 14.2.2.10.0.28 Field Documentation

##### 14.2.2.10.0.28.1 volatile enet\_bd\_struct\_t\* enet\_bd\_config::txBds

This address must always be evenly divisible by 16.

##### 14.2.2.10.0.28.2 uint8\_t\* enet\_bd\_config::txBuffer

This address must always be evenly divisible by 16.

##### 14.2.2.10.0.28.3 uint32\_t enet\_bd\_config::txBdNumber

##### 14.2.2.10.0.28.4 uint32\_t enet\_bd\_config::txBuffSizeAlign

##### 14.2.2.10.0.28.5 volatile enet\_bd\_struct\_t\* enet\_bd\_config::rxBds

This address must always be evenly divisible by 16.

##### 14.2.2.10.0.28.6 uint8\_t\* enet\_bd\_config::rxBuffer

This address must always be evenly divisible by 16.

##### 14.2.2.10.0.28.7 uint32\_t enet\_bd\_config::rxBdNumber

##### 14.2.2.10.0.28.8 uint32\_t enet\_bd\_config::rxBuffSizeAlign

#### 14.2.2.11 struct enet\_cur\_status\_t

##### Data Fields

- [enet\\_mib\\_rx\\_stat\\_t rxStatic](#)  
*The Rx static event counter.*
- [enet\\_mib\\_tx\\_stat\\_t txStatic](#)  
*The Tx static event counter.*
- [uint16\\_t maxFrameLen](#)  
*The max frame length.*
- [uint32\\_t statusFlags](#)  
*The status flag.*

#### 14.2.2.12 struct enet\_bd\_attr\_t

##### Data Fields

- [uint16\\_t bdCtl](#)  
*Buffer descriptor control field.*
- [uint16\\_t rxBdExtCtl](#)  
*Buffer descriptor extend control field.*
- [uint16\\_t rxBdExtCtl1](#)  
*Buffer descriptor extend control field 1.*
- [uint16\\_t rxBdExtCtl2](#)

## ENET HAL driver

- `uint16_t bdLen`  
*Buffer descriptor extend control field 2.*
- `uint32_t bdTimestamp`  
*Buffer descriptor data length field.*
- `uint64_t flags`  
*Buffer descriptor time stamp field.*  
*The status flag in the buffer descriptor.*

### 14.2.3 Macro Definition Documentation

#### 14.2.3.1 `#define ENET_ALIGN( x, align ) ((unsigned int)((x) + ((align)-1)) & (unsigned int)(~((unsigned int)((align)- 1)))`

Defines the alignment operation.

### 14.2.4 Enumeration Type Documentation

#### 14.2.4.1 `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_PtpringBufferFull` PTP ring buffer full.
- `kStatus_ENET_PtpringBufferEmpty` 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.  
*kStatus\_ENET\_NoMulticastAddr* No multicast group address.  
*kStatus\_ENET\_AlreadyAddedMulticast* Have Already added to multicast group.  
*kStatus\_ENET\_PHYAutoDiscoverFail* Failed to automatically discover PHY.

#### 14.2.4.2 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.

#### 14.2.4.3 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.

## ENET HAL driver

### 14.2.4.4 enum enet\_rx\_bd\_control\_extend1\_t

Enumerator

*kEnetRxBdInterrupt* BD interrupt.  
*kEnetRxBdUnicast* Unicast frame.  
*kEnetRxBdCollision* BD collision.  
*kEnetRxBdPhyErr* PHY error.  
*kEnetRxBdMacErr* Mac error.

### 14.2.4.5 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.

### 14.2.4.6 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.

### 14.2.4.7 enum enet\_tx\_bd\_control\_extend1\_t

Enumerator

*kEnetTxBdTxInterrupt* Transmit interrupt.  
*kEnetTxBdTimeStamp* Transmit timestamp flag.

#### 14.2.4.8 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.

#### 14.2.4.9 enum enet\_fifo\_configure\_t

Enumerator

- kEnetMinTxFifoAlmostFull* ENET minimum transmit FIFO almost full value.
- kEnetMinFifoAlmostEmpty* ENET minimum FIFO almost empty value.
- kEnetDefaultTxFifoAlmostFull* ENET default transmit FIFO almost full value.

#### 14.2.4.10 enum enet\_mac\_operate\_mode\_t

Enumerator

- kEnetMacNormalMode* Normal operating mode for ENET MAC.
- kEnetMacSleepMode* Sleep mode for ENET MAC.

#### 14.2.4.11 enum enet\_config\_rmii\_mode\_t

Enumerator

- kEnetCfgMii* MII mode for data interface.
- kEnetCfgRmii* RMII mode for data interface.

#### 14.2.4.12 enum enet\_config\_speed\_t

Enumerator

- kEnetCfgSpeed100M* Speed 100 M mode.
- kEnetCfgSpeed10M* Speed 10 M mode.

## ENET HAL driver

### 14.2.4.13 enum enet\_config\_duplex\_t

Enumerator

- kEnetCfgHalfDuplex* Half duplex mode.
- kEnetCfgFullDuplex* Full duplex mode.

### 14.2.4.14 enum enet\_mii\_write\_t

Enumerator

- kEnetWriteNoCompliant* Write frame operation, but not MII compliant.
- kEnetWriteValidFrame* Write frame operation for a valid MII management frame.

### 14.2.4.15 enum enet\_mii\_read\_t

Enumerator

- kEnetReadValidFrame* Read frame operation for a valid MII management frame.
- kEnetReadNoCompliant* Read frame operation, but not MII compliant.

### 14.2.4.16 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.

### 14.2.4.17 enum enet\_timer\_channel\_t

Enumerator

- kEnetTimerChannel1* IEEE 1588 timer Channel 1.
- kEnetTimerChannel2* IEEE 1588 timer Channel 2.
- kEnetTimerChannel3* IEEE 1588 timer Channel 3.
- kEnetTimerChannel4* IEEE 1588 timer Channel 4.



#### 14.2.4.18 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 IEEE 1588 clock cycle.
- kEnetChannelPulseHighonCompare* Pulse output high on compare for one IEEE 1588 clock cycle.

#### 14.2.4.19 enum enet\_interrupt\_request\_t

Enumerator

- kEnetBabrInterrupt* Babbling receive error interrupt source.
- kEnetBabtInterrupt* 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.

#### 14.2.4.20 enum enet\_irq\_number\_t

Enumerator

- kEnetTsTimerNumber* ENET ts\_timer irq number.

## ENET HAL driver

*kEnetReceiveNumber* ENET receive irq number.  
*kEnetTransmitNumber* ENET transmit irq number.  
*kEnetMiiErrorNumber* ENET mii error irq number.

### 14.2.4.21 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.

### 14.2.4.22 enum enet\_txaccelerator\_config\_t

Enumerator

*kEnetTxAccelShift16Enabled* Tx FIFO shift-16.  
*kEnetTxAccelIpChecksumEnabled* Insert IP header checksum.  
*kEnetTxAccelProtoChecksumEnabled* Insert protocol checksum.

### 14.2.4.23 enum enet\_rxaccelerator\_config\_t

Enumerator

*kEnetRxAccelPadRemoveEnabled* Padding removal for short IP frames.  
*kEnetRxAccelIpChecksumEnabled* Discard with wrong IP header checksum.  
*kEnetRxAccelProtoChecksumEnabled* Discard with wrong protocol checksum.  
*kEnetRxAccelMacChecksumEnabled* Discard with Mac layer errors.  
*kEnetRxAccelShift16Enabled* Rx FIFO shift-16.

#### 14.2.4.24 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.
- kEnetSMIPreambleDisable* Enable SMI preamble.
- kEnetVlanTagEnabled* Enable Vlan Tag.
- kEnetMacEnhancedEnable* Enable enhanced MAC feature (IEEE 1588 feature/enhanced buff descriptor)

#### 14.2.4.25 enum enet\_en\_dynamical\_act\_t

Enumerator

- kEnGraceSendStop* Enable/disable mac to stop the sending process gracefully.
- kEnSendPauseFrame* Enable/disable mac to send the pause frame after current data frame is sent.
- kEnClearMibCounter* Enable/disable mac to clear the mib counter.

### 14.2.5 Function Documentation

#### 14.2.5.1 enet\_status\_t ENET\_HAL\_Init ( ENET\_Type \* base )

Parameters

---

## ENET HAL driver

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

### Returns

The status of the initialize operation.

- false initialize failure.
- true initialize success.

**14.2.5.2 void ENET\_HAL\_Config ( ENET\_Type \* *base*, const enet\_mac\_config\_t \* *macCfgPtr*, const uint32\_t *sysClk*, const enet\_bd\_config \* *bdConfig* )**

### Parameters

|                      |                                         |
|----------------------|-----------------------------------------|
| <i>base</i>          | The ENET peripheral base address.       |
| <i>macCfgPtr</i>     | MAC controller related configuration.   |
| <i>sysClk</i>        | The system clock                        |
| <i>bufDespConfig</i> | buffer descriptor related configuration |

**14.2.5.3 void ENET\_HAL\_GetStatus ( ENET\_Type \* *base*, const uint32\_t *mask*, enet\_cur\_status\_t \* *curStatus* )**

### Parameters

|                  |                                                   |
|------------------|---------------------------------------------------|
| <i>base</i>      | The ENET peripheral base address.                 |
| <i>mask</i>      | The mask represent which status user want to get. |
| <i>curStatus</i> | The structure to save the status result           |

**14.2.5.4 void ENET\_HAL\_SetMulticastAddrHash ( ENET\_Type \* *base*, 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>base</i>     | The ENET peripheral base address.                                           |
| <i>crcValue</i> | The CRC value of the multicast group address.                               |
| <i>mode</i>     | The operation for initialize/enable/disable the specified hardware address. |

#### 14.2.5.5 void ENET\_HAL\_GetBufDescripAttr ( volatile enet\_bd\_struct\_t \* *curBd*, const uint64\_t *mask*, enet\_bd\_attr\_t \* *resultAttr* )

Parameters

|                   |                                                                   |
|-------------------|-------------------------------------------------------------------|
| <i>curBd</i>      | The ENET buffer descriptor address.                               |
| <i>mask</i>       | The attribute mask represent which field user want to get.        |
| <i>resultAttr</i> | the attribute value which is updated according to the mask value. |

#### 14.2.5.6 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.

#### 14.2.5.7 void ENET\_HAL\_ClrRxBdAfterHandled ( volatile enet\_bd\_struct\_t \* *rxBds*, uint8\_t \* *data*, bool *isbufferUpdate* )

This interface mainly clears the status region and update the buffer pointer of the rx descriptor to a null buffer to ensure that the BD is correctly available to receive data.

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 isbufferUpdate is set. |

## ENET HAL driver

|                       |                                                                                                                             |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------|
| <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. |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------|

### 14.2.5.8 void ENET\_HAL\_SetTxBdBeforeSend ( volatile enet\_bd\_struct\_t \* *txBds*, uint16\_t *length*, bool *isTxTsCfged*, bool *isTxCrcEnable*, bool *isLastOne* )

This interface mainly clears the status region of TX buffer descriptor to ensure tat the BD is correctly available to send. 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> | The flag to transmit CRC sequence after the data byte. <ul style="list-style-type: none"> <li>• 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.</li> <li>• False the transmit buffer descriptor does not transmit the CRC sequence after the data byte. if the transmit CRC forward from application.</li> </ul> |
| <i>isLastOne</i>     | The last BD flag in a frame. <ul style="list-style-type: none"> <li>• True the last BD in a frame.</li> <li>• False not the last BD in a frame.</li> </ul>                                                                                                                                                                                                                                                                                              |

### 14.2.5.9 static void ENET\_HAL\_ClrTxBdAfterSend ( volatile enet\_bd\_struct\_t \* *curBd* ) [inline], [static]

Clears the data, length, control, and status region of the transmit buffer descriptor.

Parameters

|              |                                |
|--------------|--------------------------------|
| <i>curBd</i> | The current buffer descriptor. |
|--------------|--------------------------------|

```
14.2.5.10 static void ENET_HAL_SetRxBdActive (ENET_Type * base) [inline],
 [static]
```

The buffer descriptor activation should be done after the ENET module is enabled. Otherwise, the activation fails.

## ENET HAL driver

### Parameters

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

#### 14.2.5.11 static void ENET\_HAL\_SetTxBdActive ( ENET\_Type \* *base* ) [inline], [static]

The buffer descriptor activation should be done after the ENET module is enabled. Otherwise, the activation fails.

### Parameters

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

#### 14.2.5.12 void ENET\_HAL\_SetRMII Mode ( ENET\_Type \* *base*, enet\_config\_rmii\_t \* *rmiiCfgPtr* )

### Parameters

|                   |                                               |
|-------------------|-----------------------------------------------|
| <i>base</i>       | The ENET peripheral base address.             |
| <i>rmiiCfgPtr</i> | The RMII/MII configuration structure pointer. |

#### 14.2.5.13 static uint32\_t ENET\_HAL\_GetSMIData ( ENET\_Type \* *base* ) [inline], [static]

### Parameters

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

### Returns

The data read from PHY

#### 14.2.5.14 void ENET\_HAL\_SetSMIRRead ( ENET\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *phyReg*, enet\_mii\_read\_t *operation* )



## Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>base</i>      | The ENET peripheral base address. |
| <i>phyAddr</i>   | The PHY address.                  |
| <i>phyReg</i>    | The PHY register.                 |
| <i>operation</i> | The read operation.               |

**14.2.5.15 void ENET\_HAL\_SetSMIWrite ( ENET\_Type \* *base*, uint32\_t *phyAddr*, uint32\_t *phyReg*, enet\_mii\_write\_t *operation*, uint32\_t *data* )**

## Parameters

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

**14.2.5.16 void ENET\_HAL\_EnDynamicalAct ( ENET\_Type \* *base*, enet\_en\_dynamical\_act\_t *action*, bool *enable* )**

## Parameters

|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>base</i>   | The ENET peripheral base address.                   |
| <i>action</i> | The action which will be enabled/disabled.          |
| <i>enable</i> | The switch to enable/disable the action of the MAC. |

**14.2.5.17 static void ENET\_HAL\_Enable ( ENET\_Type \* *base* ) [inline], [static]**

## Parameters

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

**14.2.5.18 static void ENET\_HAL\_Disable ( ENET\_Type \* *base* ) [inline], [static]**

## ENET HAL driver

### Parameters

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

### 14.2.5.19 void ENET\_HAL\_SetIntMode ( ENET\_Type \* *base*, enet\_interrupt\_request\_t *source*, bool *enable* )

### Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>base</i>   | The ENET peripheral base address. |
| <i>source</i> | The interrupt sources.            |
| <i>enable</i> | The interrupt enable switch.      |

### 14.2.5.20 static void ENET\_HAL\_ClearIntStatusFlag ( ENET\_Type \* *base*, enet\_interrupt\_request\_t *source* ) [inline], [static]

### Parameters

|               |                                                                                                                 |
|---------------|-----------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | The ENET peripheral base address.                                                                               |
| <i>source</i> | The interrupt source to be cleared. enet_interrupt_request_t enum types is recommended as the interrupt source. |

### 14.2.5.21 static bool ENET\_HAL\_GetIntStatusFlag ( ENET\_Type \* *base*, enet\_interrupt\_request\_t *source* ) [inline], [static]

### Parameters

|               |                                                                                                    |
|---------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>   | The ENET peripheral base address.                                                                  |
| <i>source</i> | The interrupt sources. enet_interrupt_request_t 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.

**14.2.5.22 void ENET\_HAL\_Start1588Timer ( ENET\_Type \* *base*, enet\_config\_ptp\_timer\_t \* *ptpCfgPtr* )**

This interface configures the IEEE 1588 timer and starts the IEEE 1588 timer. After the timer starts the IEEE 1588 timer starts incrementing.

## ENET HAL driver

### Parameters

|                  |                                                      |
|------------------|------------------------------------------------------|
| <i>base</i>      | The ENET peripheral base address.                    |
| <i>ptpCfgPtr</i> | The IEEE 1588 timer configuration structure pointer. |

### 14.2.5.23 void ENET\_HAL\_Stop1588Timer ( ENET\_Type \* *base* )

This interface stop the IEEE 1588 timer and clear its count value.

### Parameters

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

### 14.2.5.24 static void ENET\_HAL\_Adjust1588Timer ( ENET\_Type \* *base*, uint32\_t *increaseCorrection*, uint32\_t *periodCorrection* ) [inline], [static]

Adjust the IEEE 1588 timer according to the increase and correction period of the configured correction.

### Parameters

|                            |                                              |
|----------------------------|----------------------------------------------|
| <i>base</i>                | The ENET peripheral base address.            |
| <i>increase-Correction</i> | The increase correction for IEEE 1588 timer. |
| <i>period-Correction</i>   | The period correction for IEEE 1588 timer.   |

### 14.2.5.25 static void ENET\_HAL\_Set1588TimerNewTime ( ENET\_Type \* *base*, uint32\_t *nanSecond* ) [inline], [static]

### Parameters

|                  |                                        |
|------------------|----------------------------------------|
| <i>base</i>      | The ENET peripheral base address.      |
| <i>nanSecond</i> | The nanosecond set to IEEE 1588 timer. |

### 14.2.5.26 static uint32\_t ENET\_HAL\_Get1588TimerCurrentTime ( ENET\_Type \* *base* ) [inline], [static]

Sets the capture command to the IEEE 1588 timer is used before reading the current time register. After set timer capture, please wait for about 1us before read the captured timer.

## Parameters

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

## Returns

the current time from IEEE 1588 timer.

**14.2.5.27** `static bool ENET_HAL_Get1588TimerChnStatus ( ENET_Type * base,  
enet_timer_channel_t channel ) [inline], [static]`

## Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>base</i>    | The ENET peripheral base address.   |
| <i>channel</i> | The IEEE 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.

**14.2.5.28** `static void ENET_HAL_Rst1588TimerCmpValAndClrFlag ( ENET_Type *  
base, enet_timer_channel_t channel, uint32_t compareValue ) [inline],  
[static]`

## Parameters

|                     |                                            |
|---------------------|--------------------------------------------|
| <i>base</i>         | The ENET peripheral base address.          |
| <i>channel</i>      | The IEEE 1588 timer channel number.        |
| <i>compareValue</i> | Compare value for IEEE 1588 timer channel. |

---

## ENET RTCS Adaptor

### 14.3 ENET RTCS Adaptor

This chapter describes the programming interface of the ENET RTCS Adaptor.

## **14.4 ENET Physical Layer Driver**

This chapter describes the programming interface of the ENET Physical Layer Driver.

## ENET Peripheral Driver

### 14.5 ENET Peripheral Driver

#### 14.5.1 Overview

This section describes the programming interface of the ENET Peripheral Driver. The ENET driver receives data from and transmits data to the wired network. The enhanced 1588 feature supports clock synchronization.

#### 14.5.2 ENET device data structure

The ENET device data structure, [enet\\_dev\\_if\\_t](#), includes configuration structure, application structure, and data context structure. This structure should be initialized before the ENET device initialization function. Then, the data structure is passed from the API layer to the device.

#### 14.5.3 ENET Configuration structure

The ENET device data structure uses the [enet\\_mac\\_config\\_t](#) and the [enet\\_phy\\_config\\_t](#) configuration structure for MAC and PHY configurations. This allows users to configure the most common settings of the ENET peripheral. The [enet\\_mac\\_config\\_t](#) mac configuration structure includes the Mac mode, MI/RMII mode configuration, MAC controller configuration, receive and transmit accelerator, receive and transmit FIFO, special Mac configuration to receive maximum frame length, receive truncate length, pause duration for pause frame and the PTP slave mode. The [enet\\_phy\\_config\\_t](#) PHY configuration structure includes the PHY address and PHY loop mode, which needs to be configured. If the PHY address is unknown, use the `isAutodiscoverEnabled` flag to find the PHY address. Note:

1. The default maximum frame length is 1518 or 1522(VLAN). The default IPG (inter-packet-gap) is 12 bytes, a sufficient amount for normal Ethernet use. If no special requirements are present, let the `macSpecialCfg` be NULL.
2. The recommended receive and transmit buffer size is the maximum frame size: either 1518 or 1522(VLAN).
3. If the `kEnetTxAccelEnable` and `kEnetRxAccelEnable` are set, ensure that the `txAccelerCfg` and `rxAccelerCfg` are configured.

#### 14.5.4 ENET Buffer data structure

The ENET device data structure uses the [enet\\_buff\\_config\\_t](#) to configure the receive/transmit buffer descriptor numbers and address, the receive/transmit data buffer, the aligned receive/transmit buffer size, the extended receive data buffer queue address and 1588 PTP timestamp data buffer address, etc. Data receive and transmit is easily managed with this context structure.



## 14.5.5 ENET Initialization

To initialize the ENET module, follow these steps:

1. First, initialize the ENET MAC and PHY configuration structures, and initialize the upper layer callback function for interrupt mode or initialize semaphore `enetIfPtr->enetReceiveSync`, create receive task for poll mode.
2. Call the `enet_mac_init()` function with the `enet_mac_api_t` API structure in the `enet_dev_if_t` and pass in the `enet_dev_if_t` device data structure. This function enables the ENET module.

This is an example code to initialize the ENET device data structure:

```

uint32_t phyAddr;
enet_mac_config_t configMac;
enet_buff_config_t bufferCfg;
enet_config_rmii_t rmiiCfg;
rmiiCfg.duplex = kEnetCfgFullDuplex;
rmiiCfg.speed = kEnetCfgSpeed100M;
rmiiCfg.mode = kEnetCfgRmii;
rmiiCfg.isLoopEnabled = false;
rmiiCfg.isRxOnTxDisabled = false;
memset(&configMac, 0, sizeof(enet_mac_config_t));
configMac.macMode = kEnetMacNormalMode;
configMac.macAddr = source;
configMac.rmiiCfgPtr = &rmiiCfg;
configMac.macCtlConfigure = kEnetRxCrcFwdEnable |
 kEnetTxCrcBdEnable | kEnetMacEnhancedEnable;

// Buffer configure structure//

memset(&bufferCfg, 0, sizeof(enet_buff_config_t));
bufferCfg.rxBdNumber = ENET_RXBD_NUM;
bufferCfg.rxBdPtrAlign = RxBuffDescrip;
bufferCfg.rxBufferAlign = &RxDataBuff[0][0];
bufferCfg.rxBuffSizeAlign = ENET_RXBuffSizeAlign(ENET_RXBUFF_SIZE);
bufferCfg.txBdNumber = ENET_TXBD_NUM;
bufferCfg.txBdPtrAlign = TxBuffDescrip;
bufferCfg.txBufferAlign = &TxDataBuff[0][0];
bufferCfg.txBuffSizeAlign = ENET_TXBuffSizeAlign(ENET_TXBUFF_SIZE);
#if !ENET_RECEIVE_ALL_INTERRUPT
bufferCfg.extRxBuffQue = &ExtRxDataBuff[0][0];
bufferCfg.extRxBuffNum = ENET_EXTRXBD_NUM;
#endif
#if FSL_FEATURE_ENET_SUPPORT_PTP
configMac.isSlaveMode = false;
bufferCfg.ptpTsRxDataPtr = &ptpTsRxData[0];
bufferCfg.ptpTsRxBuffNum = ENET_PTP_RXTS_RING_LEN;
bufferCfg.ptpTsTxDataPtr = &ptpTsTxData[0];
bufferCfg.ptpTsTxBuffNum = ENET_PTP_TXTS_RING_LEN;
#endif

// Set up the PHY configuration structure//

enet_phy_config_t g_enetPhyCfg =
{{0, false }};

// Initialize ENET device//
result = ENET_DRV_Init(enetIfPtr, &configMac, &bufferCfg);
// Initialize PHY//

if(g_enetPhyCfg[device].isAutodiscoverEnabled)

```

## ENET Peripheral Driver

```

{
 uint32_t phyAddr;
 result = PHY_DRV_Autodiscover(device, &phyAddr);
 if(result != kStatus_ENET_Success)
 return result;
 enetIfPtr->phyAddr = phyAddr;
}
else
{
 enetIfPtr->phyAddr = g_enetPhyCfg[device].phyAddr;
}

PHY_DRV_Init(device, enetIfPtr->phyAddr, g_enetPhyCfg[device].isLoopEnabled);

// Install ENET stack net interface callback function for receive interrupt mode//

#ifdef ENET_RECEIVE_ALL_INTERRUPT
 ENET_DRV_InstallNetIfCall(enetIfPtr, enet_receive_test);
#else

// Initialize semaphore enetIfPtr->enetReceiveSync and create receive task for poll mode //

osa_status_t osaFlag;
osaFlag = OSA_EventCreate(&enetIfPtr->enetReceiveSync,
 kEventAutoClear);
if(osaFlag != kStatus_OSA_Success)
{
 return osaFlag;
}

/* Create receive task*/

osaFlag = OSA_TaskCreate(ENET_receive, "receive", ENET_TASK_STACK_SIZE,
 ENET_receive_stack, ENET_RECEIVE_TASK_PRIO, (task_param_t)enetIfPtr, false, &ENET_receive_task_handler);
if(osaFlag != kStatus_OSA_Success)
{
 return osaFlag;
}
}
#endif

```

### 14.5.6 ENET Data Receive

ENET can receive data in two different ways. The MACRO ENET\_RECEIVE\_ALL\_INTERRUPT selects the way in which data is received.

- interrupt and poll (define ENET\_RECEIVE\_ALL\_INTERRUPT with 0)
- interrupt only (define ENET\_RECEIVE\_ALL\_INTERRUPT with 1)

Poll approach (Interrupt add task poll) for Mac receive: The ENET driver receives data directly from the buffer descriptor to the upper layer. To shorten the receiving process time on the receive interrupt, the receive data uses the interrupt and poll combination:

1. Receive interrupt: releases the receive synchronize signal (enetReceiveSync).
2. Poll: receives task and waits for the receive synchronize signal when no data is received. The receive data returns the address and data length of the received data. To receive data from the ENET device, create a receive task as follows:

```

// Create the task when do net device initialize//

task_create(ENET_receive, ENET_TEST_TASK_PRIO, enetIfPtr, &revHandle);

```

```

.....

ENET_receive(void *param)
{
 uint8_t *packet;
 uint16_t length;
 uint16_t counter;
 enet_mac_packet_buffer_t packetBuffer[
kEnetMaxFrameBdNumbers];

 enet_dev_if_t * enetIfPtr = (enet_dev_if_t *)param;
 while(1)
 {

 // Receive frame//

 result = ENET_DRV_ReceiveData(enetIfPtr, packetBuffer);
 if ((result == kStatus_ENET_RxbdEmpty) || (result ==
kStatus_ENET_InvalidInput))
#ifdef !USE_RTOS
 {
 status = OSA_EventWait (&enetIfPtr->enetReceiveSync, flag, false, 0, &flagCheck);
 }
 else if(result == kStatus_ENET_Success)
 {
#else
 {
 OSA_EventWait (&enetIfPtr->enetReceiveSync, flag, false,
OSA_WAIT_FOREVER, &flagCheck);
 }
#endif
 }

 // The packets delivery to upper layer and do data buffer enqueue after the data copied to the
upper layer buffer//

 if (packetBuffer[0].data != NULL)
 {

 // Collect the frame first to the packet which is the data buffer for upper layer//

 length = 0;
 for(counter = 0; packetBuffer[counter].length != 0 ; counter ++)
 {
 memcpy(packet + length, packetBuffer[counter].data, packetBuffer[counter].length);
 length += packetBuffer[counter].length;
 *(uint32_t *) (packetBuffer[counter].data) = 0;

 // data buffers are required to enqueued to the extended receive buffer queue//

 enet_mac_enqueue_buffer((void **)&enetIfPtr->
bdContext.extRxBuffQue, packetBuffer[counter].data);
 packetBuffer[counter].length = 0;
 }

 // call upper layer receive function//

 }
}
</code>

```

## ENET Peripheral Driver

```

// Receive interrupt handler to wake up blocked receive task//
<code>
void ENET_DRV_RxIRQHandler(uint32_t instance)
{
 enet_dev_if_t *enetIfPtr;
 uint32_t baseAddr;
 enetIfPtr = enetIfHandle[instance];
 event_flags_t flag = 0x1;

 //Check input parameter//

 if (!enetIfPtr)
 {
 return;
 }
 baseAddr = g_enetBaseAddr[enetIfPtr->deviceNumber];

 // Get interrupt status.//

 while ((ENET_HAL_GetIntStatusFlag(baseAddr,
 kEnetRxFrameInterrupt)) || (ENET_HAL_GetIntStatusFlag(
 baseAddr, kEnetRxByteInterrupt)))
 {

 //Clear interrupt//

 ENET_HAL_ClearIntStatusFlag(baseAddr,
 kEnetRxFrameInterrupt);
 ENET_HAL_ClearIntStatusFlag(baseAddr,
 kEnetRxByteInterrupt);

 // Release sync signal-----//

 OSA_EventSet(&enetIfPtr->enetReceiveSync, flag);
 }
}

```

Note: The extended receive buffer queue is required in the structure when using the receive polling mode. The data buffer in receive polling mode has to be enqueued by the end of each receive time.

Interrupt only approach for Mac receive: The ENET driver receives data directly from the buffer descriptor to the upper layer. In this case, data is received on the receive interrupt handler:

1. Receive interrupt handler calls the receive peripheral driver.
2. Receive peripheral driver calls the initialized callback function.
3. The callback function checks the protocol and delivers the received data to the upper layer of the TCP/IP stack.

These are the details:

```

void ENET_DRV_RxIRQHandler(uint32_t instance)
{
 enet_dev_if_t *enetIfPtr;
 uint32_t baseAddr;
 enetIfPtr = enetIfHandle[instance];

 //Check input parameter//

 if (!enetIfPtr)
 {
 return;
 }
 baseAddr = g_enetBaseAddr[enetIfPtr->deviceNumber];

```

```

// Get interrupt status.//

while ((ENET_HAL_GetIntStatusFlag(baseAddr,
 kEnetRxFrameInterrupt)) || (ENET_HAL_GetIntStatusFlag(
 baseAddr, kEnetRxByteInterrupt)))
{
 //Clear interrupt//

 ENET_HAL_ClearIntStatusFlag(baseAddr,
 kEnetRxFrameInterrupt);
 ENET_HAL_ClearIntStatusFlag(baseAddr,
 kEnetRxByteInterrupt);

 // Receive peripheral driver//

 ENET_DRV_ReceiveData(enetIfPtr);
}

}

</code>
.....

<code>
uint32_t ENET_DRV_ReceiveData(enet_dev_if_t *enetIfPtr)
{
 void *curBd;
 uint32_t length;
 uint8_t *packet;
 uint32_t controlStatus;

 // Check input parameters//

 if(!enetIfPtr)
 {
 return kStatus_ENET_InvalidInput;
 }

 // callback function to delivery to stack upper layer //

 enetIfPtr->enetNetifcall(enetIfPtr, packet, length);

 return kStatus_ENET_Success;
}

</code>

<code>
uint32_t void enet_callback(void *param, enet_mac_packet_buffer_t *packetBuffer)
{
 uint32_t length = 0;
 uint16_t type, counter = 0;
 uint8_t *packet;

 // Collect the frame first//

 if(!packetBuffer[1].length)
 {
 packet = packetBuffer[0].data;
 length = packetBuffer[0].length;
 }
}

```

## ENET Peripheral Driver

```

}
else
{
 // Dequeue a large buffer //

 packet = enet_mac_dequeue_buffer((void **)&dataBuffQue);
 if(packet!=NULL)
 {
 for(counter = 0; packetBuffer[counter].next != NULL ; counter ++)
 {
 memcpy(packet + length, packetBuffer[counter].data, packetBuffer[counter].length);
 length += packetBuffer[counter].length;
 }
 }
 else
 {
 return kStatus_ENET_LargeBufferFull;
 }
}

// Process the received frame//
type = ntohs(*(uint16_t *)&((enet_etherent_header_t *)packet)->type);
// Collect frame to PCB structure for upper layer process//

QUEUEGET(packbuffer[enetIfPtr->deviceNumber].pcbHead,packbuffer[enetIfPtr->
deviceNumber].pcbTail, pcbPtr);
if(pcbPtr)
{
 pcbPtr->FRAG[0].LENGTH = length;
 pcbPtr->FRAG[0].FRAGMENT = packet;
 pcbPtr->PRIVATE = (void *)enetIfPtr;

 switch(type)
 {
 case ENETPROT_IP:
 IPE_rcv_IP((PCB *)pcbPtr,enetIfPtr->netIfPtr);
 break;
 case ENETPROT_ARP:
 IPE_rcv_ARP((PCB *)pcbPtr,enetIfPtr->netIfPtr);
 break;
 case ENETPROT_IP6:
 IP6E_rcv_IP((PCB *)pcbPtr,enetIfPtr->netIfPtr);
 break;
 case ENETPROT_ETHERNET:
 enet_ptp_service_l2packet(enetIfPtr, packet, length);
 break;
 default:
 PCB_free((PCB *)pcbPtr);
 break;
 }
}
else
{
 enetIfPtr->stats.statsRxMissed++;
}
}

```

Remember to enqueue the packet to the dataBuffQue and enqueue the pcbPtr to the packbuffer after the data is processed by the upper layer. This process is done in the ENET\_free API and called by the RTCS.  
 //

## 14.5.7 ENET Data Transmit

Data Transmit transmits data from the TCP/IP layer to the device and, then, to the network. The data transmit function should be used as follows: PCB\_PTR is the data buffer structure of the frame which contains many PCB\_FRAGMENT(including the data buffer and length).

```
uint32_t ENET_send(_enet_handle handle, PCB_PTR packet, uint32_t type, _enet_address dest, uint32_t flags)
{
 uint8_t headerLen, *frame;
 PCB_FRAGMENT *fragPtr;
 uint16_t size = 0, lenTemp = 0, bdNumUsed = 0;
 enet_dev_if_t *enetIfPtr;
 enet_ethernet_header_t *packetPtr;
 volatile enet_bd_struct_t * curBd;
 uint32_t result, lenoffset = 0;

 //Check out//

 if ((!handle) || (!packet))
 {
 return kStatus_ENET_InvalidInput;
 }

 enetIfPtr = (enet_dev_if_t *)handle;

 // Default frame header size//
 headerLen = kEnetEthernetHeadLen;

 // Check the frame length//

 for (fragPtr = packet->FRAG; fragPtr->LENGTH; fragPtr++)
 {
 size += fragPtr->LENGTH;
 }
 if (size > enetIfPtr->maxFrameSize)
 {
 return kStatus_ENET_TxLarge;
 }

 //Add MAC hardware address//

 packetPtr = (enet_ethernet_header_t *)packet->FRAG[0].FRAGMENT;
 htons(packetPtr->destAddr, dest);
 htons(packetPtr->sourceAddr, enetIfPtr->macAddr);
 packetPtr->type = htons(type);
 if (flags & ENET_OPT_8021QTAG)
 {
 enet_8021vlan_header_t *vlanHeadPtr = (
 enet_8021vlan_header_t *)packetPtr;
 vlanHeadPtr->tpidtag = htons(ENETPROT_8021Q);
 vlanHeadPtr->othertag = htons((ENET_GETOPT_8021QPRIORIO(flags) << 13));
 vlanHeadPtr->type = htons(type);
 headerLen = sizeof(enet_8021vlan_header_t);
 packet->FRAG[0].LENGTH = headerLen;
 }

 if (flags & ENET_OPT_8023)
 {
 enet_8022_header_ptr lcPtr = (enet_8022_header_ptr)(packetPtr->type + 2);
 packetPtr->type = htons(size - headerLen);
 lcPtr->dsap[0] = 0xAA;
 lcPtr->ssap[0] = 0xAA;
 lcPtr->command[0] = 0x03;
 lcPtr->oui[0] = 0x00;
 }
}
```

## ENET Peripheral Driver

```

 lcPtr->oui[1] = 0x00;
 lcPtr->oui[2] = 0x00;
 lcPtr->type = HTONS(type);
 packet->FRAG[0].LENGTH = packet->FRAG[0].LENGTH+ sizeof(enet_8022_header_t);
}

// Get the current transmit data buffer in buffer descriptor //

curBd = enetIfPtr->bdContext.txBdCurPtr;
frame = ENET_HAL_GetBuffDescripData(curBd);

// Send a whole frame with a signal buffer//

if(size <= enetIfPtr->bdContext.txBuffSizeAlign)
{
 bdNumUsed = 1;
 for (fragPtr = packet->FRAG; fragPtr->LENGTH; fragPtr++)
 {
 memcpy(frame + lenTemp, fragPtr->FRAGMENT, fragPtr->LENGTH);
 lenTemp += fragPtr->LENGTH;
 }

 // Send packet to the device//

 return ENET_DRV_SendData(enetIfPtr, size, bdNumUsed);
}

// Copy the Ethernet header first//
memcpy(frame, packet->FRAG[0].FRAGMENT, packet->FRAG[0].LENGTH);

// Send a whole frame with multiple buffer descriptors//

while((size - bdNumUsed* enetIfPtr->bdContext.txBuffSizeAlign) > enetIfPtr->
 bdContext.txBuffSizeAlign)
{
 if(bdNumUsed == 0)
 {
 memcpy((void *) (frame + packet->FRAG[0].LENGTH), (void *) (packet->FRAG[1].FRAGMENT), enetIfPtr
->bdContext.txBuffSizeAlign - packet->FRAG[0].LENGTH);
 lenoffset += (enetIfPtr->bdContext.txBuffSizeAlign - packet->FRAG[0].
LENGTH);
 }
 else
 {
 memcpy((void *) frame, (void *) (packet->FRAG[1].FRAGMENT + lenoffset), enetIfPtr->
bdContext.txBuffSizeAlign);
 lenoffset += enetIfPtr->bdContext.txBuffSizeAlign;
 }

 // Increment the buffer descriptor//

 curBd = ENET_DRV_IncrTxBuffDescripIndex(enetIfPtr, curBd);
 frame = ENET_HAL_GetBuffDescripData(curBd);

 // Increment the index and parameters//

 bdNumUsed ++;
}
memcpy((void *) frame, (void *) (packet->FRAG[1].FRAGMENT + lenoffset), size - bdNumUsed * enetIfPtr->
 bdContext.txBuffSizeAlign);
bdNumUsed ++;

// Send packet to the device//

result = ENET_DRV_SendData(enetIfPtr, size, bdNumUsed);

// Free the PCB buffer if necessary//

```



```
PCB_free(packet);
return result;
}
```

### 14.5.8 ENET Note:

- TWR-MK70F120M Tower System module routes the RMIi interface signals from the CPU board to the primary connectors. Hence, the `rmiiCfgMode` in the `enet_mac_config_t` should be set to the `kEnetCfgRmii`. TWR-MK64F120M module can configure both RMIi and MII modes.
- Jumper settings When ENET is uses the RMIi interface signals by default, the RMIi input clock must be kept in phase with the clock supplied to the external PHY. Jumpers should be set as follows:
  - TWR-SER J2 shunt across 3-4 , J3 shunt across 2-3, J12 shunt across 9-10
  - TWR-MK64f120M board make sure J32 jumper is on to disable the OSC on the CPU board.
  - TWR-MK70f120M board make sure J18 jumper is on to disable the OSC on the CPU board.
 When ENET chooses the MII interface signals, the jumper should be set as follows: TWR-SER J2 shunt across 1-2, J12 no shunt across 9-10.

### Data Structures

- struct `enet_multicast_group_t`  
*Defines the multicast group structure for the ENET device. [More...](#)*
- struct `enet_ethernet_header_t`  
*Defines the ENET header structure. [More...](#)*
- struct `enet_8021vlan_header_t`  
*Defines the ENET VLAN frame header structure. [More...](#)*
- struct `enet_buff_descrip_context_t`  
*Defines the structure for ENET buffer descriptors status. [More...](#)*
- struct `enet_stats_t`  
*Defines the ENET packets statistic structure. [More...](#)*
- struct `enet_mac_packet_buffer_t`  
*Defines the ENET MAC packet buffer structure. [More...](#)*
- struct `enet_buff_config_t`  
*Defines the receive buffer descriptor configure structure. [More...](#)*
- struct `enet_dev_if_t`  
*Defines the ENET device data structure for the ENET. [More...](#)*
- struct `enet_user_config_t`  
*Defines the ENET user configuration structure. [More...](#)*

### Macros

- `#define ENET_RECEIVE_ALL_INTERRUPT 1`  
*Defines the approach: ENET interrupt handler receive.*
- `#define ENET_ENABLE_DETAIL_STATS 0`  
*Defines the statistic enable macro.*

## ENET Peripheral Driver

- #define `ENET_HTONS(n) __REV16(n)`  
*brief Defines the macro for converting constants from host byte order to network byte order*
- #define `ENET_ORIGINAL_CRC32 0xFFFFFFFFU`  
*Defines the CRC-32 calculation constant.*
- #define `ENET_CRC32_POLYNOMIC 0xEDB88320U`  
*CRC-32 Polynomic.*

## Enumerations

- enum `enet_crc_parameter_t` {  
`kEnetCrcOffset = 8,`  
`kEnetCrcMask1 = 0x3F` }  
*Defines the CRC data for a hash value calculation.*
- enum `enet_protocol_type_t` {  
`kEnetProtocolI2ptpv2 = 0x88F7,`  
`kEnetProtocolIpv4 = 0x0800,`  
`kEnetProtocolIpv6 = 0x86dd,`  
`kEnetProtocol8021QVlan = 0x8100,`  
`kEnetPacketUdpVersion = 0x11,`  
`kEnetPacketIpv4Version = 0x4,`  
`kEnetPacketIpv6Version = 0x6` }  
*Defines the ENET protocol type and main parameters.*

## Variables

- `ENET_Type *const g_enetBase []`  
*Array for ENET module register base address.*
- `const IRQn_Type g_enetTxIrqId []`  
*Two-dimensional array for the ENET interrupt vector number.*

## ENET Driver

- `enet_status_t ENET_DRV_Init (enet_dev_if_t *enetIfPtr, const enet_user_config_t *userConfig)`  
*Initializes the ENET with the basic configuration.*
- `enet_status_t ENET_DRV_Deinit (enet_dev_if_t *enetIfPtr)`  
*De-initializes the ENET device.*
- `enet_status_t ENET_DRV_UpdateRxBuffDescrip (enet_dev_if_t *enetIfPtr, bool isBuffUpdate)`  
*Updates the receive buffer descriptor.*
- `enet_status_t ENET_DRV_CleanupTxBuffDescrip (enet_dev_if_t *enetIfPtr)`  
*ENET transmit buffer descriptor cleanup.*
- `volatile enet_bd_struct_t * ENET_DRV_IncRxBuffDescripIndex (enet_dev_if_t *enetIfPtr, volatile enet_bd_struct_t *curBd)`  
*Increases the receive buffer descriptor to the next one.*
- `volatile enet_bd_struct_t * ENET_DRV_IncTxBuffDescripIndex (enet_dev_if_t *enetIfPtr, volatile enet_bd_struct_t *curBd)`

- Increases the transmit buffer descriptor to the next one.*

  - bool [ENET\\_DRV\\_RxErrorStats](#) ([enet\\_dev\\_if\\_t](#) \*enetIfPtr, volatile [enet\\_bd\\_struct\\_t](#) \*curBd)

*Processes the ENET receive frame error statistics.*
- void [ENET\\_DRV\\_TxErrorStats](#) ([enet\\_dev\\_if\\_t](#) \*enetIfPtr, volatile [enet\\_bd\\_struct\\_t](#) \*curBd)

*Processes the ENET transmit frame statistics.*
- [enet\\_status\\_t](#) [ENET\\_DRV\\_ReceiveData](#) ([enet\\_dev\\_if\\_t](#) \*enetIfPtr)

*Receives ENET packets.*
- [enet\\_status\\_t](#) [ENET\\_DRV\\_InstallNetIfCall](#) ([enet\\_dev\\_if\\_t](#) \*enetIfPtr, [enet\\_netif\\_callback\\_t](#) function)

*Installs ENET TCP/IP stack net interface callback function.*
- [enet\\_status\\_t](#) [ENET\\_DRV\\_SendData](#) ([enet\\_dev\\_if\\_t](#) \*enetIfPtr, [uint32\\_t](#) dataLen, [uint32\\_t](#) bdNum-Used)

*Transmits ENET packets.*
- void [ENET\\_DRV\\_RxIRQHandler](#) ([uint32\\_t](#) instance)

*The ENET receive interrupt handler.*
- void [ENET\\_DRV\\_TxIRQHandler](#) ([uint32\\_t](#) instance)

*The ENET transmit interrupt handler.*
- void [ENET\\_DRV\\_CalculateCrc32](#) ([uint8\\_t](#) \*address, [uint32\\_t](#) \*crcValue)

*Calculates the CRC hash value.*
- [enet\\_status\\_t](#) [ENET\\_DRV\\_AddMulticastGroup](#) ([uint32\\_t](#) instance, [uint8\\_t](#) \*address, [uint32\\_t](#) \*hash)

*Adds the ENET device to a multicast group.*
- [enet\\_status\\_t](#) [ENET\\_DRV\\_LeaveMulticastGroup](#) ([uint32\\_t](#) instance, [uint8\\_t](#) \*address)

*Moves the ENET device from a multicast group.*
- void [enet\\_mac\\_enqueue\\_buffer](#) (void \*\*queue, void \*buffer)

*ENET buffer enqueue.*
- void \* [enet\\_mac\\_dequeue\\_buffer](#) (void \*\*queue)

*ENET buffer dequeue.*

### 14.5.9 Data Structure Documentation

#### 14.5.9.1 struct enet\_multicast\_group\_t

##### Data Fields

- [uint8\\_t](#) [groupAddr](#) [[kEnetMacAddrLen](#)]
- Multicast group address.*
- [uint32\\_t](#) [hash](#)
- Hash value of the multicast group address.*
- struct [ENETMulticastGroup](#) \* [next](#)
- Pointer of the next group structure.*
- struct [ENETMulticastGroup](#) \* [prv](#)
- Pointer of the previous structure.*

## ENET Peripheral Driver

### 14.5.9.2 struct enet\_ethernet\_header\_t

#### Data Fields

- uint8\_t **destAddr** [kEnetMacAddrLen]  
*Destination address.*
- uint8\_t **sourceAddr** [kEnetMacAddrLen]  
*Source address.*
- uint16\_t **type**  
*Protocol type.*

### 14.5.9.3 struct enet\_8021vlan\_header\_t

#### Data Fields

- uint8\_t **destAddr** [kEnetMacAddrLen]  
*Destination address.*
- uint8\_t **sourceAddr** [kEnetMacAddrLen]  
*Source address.*
- uint16\_t **tpidtag**  
*ENET 8021tag header tag region.*
- uint16\_t **othertag**  
*ENET 8021tag header type region.*
- uint16\_t **type**  
*Protocol type.*

### 14.5.9.4 struct enet\_buff\_descrip\_context\_t

#### Data Fields

- volatile **enet\_bd\_struct\_t \* rxBdBasePtr**  
*Receive buffer descriptor base address pointer.*
- volatile **enet\_bd\_struct\_t \* rxBdCurPtr**  
*Current receive buffer descriptor pointer.*
- volatile **enet\_bd\_struct\_t \* rxBdDirtyPtr**  
*Receive dirty buffer descriptor.*
- volatile **enet\_bd\_struct\_t \* txBdBasePtr**  
*Transmit buffer descriptor base address pointer.*
- volatile **enet\_bd\_struct\_t \* txBdCurPtr**  
*Current transmit buffer descriptor pointer.*
- volatile **enet\_bd\_struct\_t \* txBdDirtyPtr**  
*Last cleaned transmit buffer descriptor pointer.*
- bool **isTxBdFull**  
*Transmit buffer descriptor full.*
- bool **isRxBdFull**  
*Receive buffer descriptor full.*
- uint32\_t **rxBuffSizeAlign**  
*Receive buffer size alignment.*
- uint32\_t **txBuffSizeAlign**

- *Transmit buffer size alignment.*
- `uint8_t * extRxBuffQue`  
*Extended Rx data buffer queue to update the data buff in the receive buffer descriptor.*
- `uint8_t extRxBuffNum`  
*extended data buffer number*

#### 14.5.9.5 struct enet\_stats\_t

##### Data Fields

- `uint32_t statsRxTotal`  
*Total number of receive packets.*
- `uint32_t statsTxTotal`  
*Total number of transmit packets.*

#### 14.5.9.6 struct enet\_mac\_packet\_buffer\_t

##### Data Fields

- `uint8_t * data`  
*Data buffer pointer.*
- `uint16_t length`  
*Data length.*
- `struct ENETMacPacketBuffer * next`  
*Next pointer.*

#### 14.5.9.7 struct enet\_buff\_config\_t

##### Data Fields

- `uint16_t rxBdNumber`  
*Receive buffer descriptor number.*
- `uint16_t txBdNumber`  
*Transmit buffer descriptor number.*
- `uint32_t rxBuffSizeAlign`  
*Aligned receive buffer size and must be larger than 256.*
- `uint32_t txBuffSizeAlign`  
*Aligned transmit buffer size and must be larger than 256.*
- `volatile enet_bd_struct_t * rxBdPtrAlign`  
*Aligned receive buffer descriptor pointer.*
- `uint8_t * rxBufferAlign`  
*Aligned receive data buffer pointer.*
- `volatile enet_bd_struct_t * txBdPtrAlign`  
*Aligned transmit buffer descriptor pointer.*
- `uint8_t * txBufferAlign`  
*Aligned transmit buffer descriptor pointer.*
- `uint8_t * extRxBuffQue`  
*Extended Rx data buffer queue to update the data buff in the receive buffer descriptor.*

## ENET Peripheral Driver

- `uint8_t extRxBuffNum`  
*extended data buffer number*

### 14.5.9.8 struct enet\_dev\_if\_t

#### Data Fields

- `struct ENETDevIf * next`  
*Next device structure address.*
- `void * netIfPrivate`  
*For upper layer private structure.*
- `enet_multicast_group_t * multiGroupPtr`  
*Multicast group chain.*
- `uint32_t deviceNumber`  
*Device number.*
- `uint8_t macAddr [kEnetMacAddrLen]`  
*Mac address.*
- `uint8_t phyAddr`  
*PHY address connected to this device.*
- `bool isInitialized`  
*Device initialized.*
- `uint16_t maxFrameSize`  
*Mac maximum frame size.*
- `bool isVlanTagEnabled`  
*ENET VLAN-TAG frames enabled.*
- `bool isTxCrcEnable`  
*Transmit CRC enable in buffer descriptor.*
- `bool isRxCrcFwdEnable`  
*Receive CRC forward.*
- `enet_buff_descrip_context_t bdContext`  
*Mac buffer descriptors context pointer.*
- `enet_stats_t stats`  
*Packets statistic.*
- `enet_netif_callback_t enetNetifcall`  
*Receive callback function to the upper layer.*
- `mutex_t enetContextSync`  
*Sync signal.*

### 14.5.9.9 struct enet\_user\_config\_t

#### Data Fields

- `const enet_mac_config_t * macCfgPtr`  
*MAC configuration structure.*
- `const enet_buff_config_t * buffCfgPtr`  
*ENET buffer configuration structure.*

## 14.5.10 Macro Definition Documentation

14.5.10.1 **#define ENET\_ENABLE\_DETAIL\_STATS 0**

14.5.10.2 **#define ENET\_ORIGINAL\_CRC32 0xFFFFFFFFU**

CRC-32 Original data

## 14.5.11 Enumeration Type Documentation

14.5.11.1 **enum enet\_crc\_parameter\_t**

Enumerator

*kEnetCrcOffset* CRC-32 offset2.

*kEnetCrcMask1* CRC-32 mask.

14.5.11.2 **enum enet\_protocol\_type\_t**

Enumerator

*kEnetProtocoll2ptpv2* Packet type Ethernet ieee802.3.

*kEnetProtocollpv4* Packet type IPv4.

*kEnetProtocollpv6* Packet type IPv6.

*kEnetProtocol8021QVlan* Packet type VLAN.

*kEnetPacketUdpVersion* UDP protocol type.

*kEnetPacketIpv4Version* Packet IP version IPv4.

*kEnetPacketIpv6Version* Packet IP version IPv6.

## 14.5.12 Function Documentation

14.5.12.1 **enet\_status\_t ENET\_DRV\_Init ( enet\_dev\_if\_t \* *enetIfPtr*, const enet\_user\_config\_t \* *userConfig* )**

Parameters

|                  |                                              |
|------------------|----------------------------------------------|
| <i>enetIfPtr</i> | The pointer to the basic Ethernet structure. |
|------------------|----------------------------------------------|

## ENET Peripheral Driver

|                   |                                                |
|-------------------|------------------------------------------------|
| <i>userConfig</i> | The ENET user configuration structure pointer. |
|-------------------|------------------------------------------------|

Returns

The execution status.

### 14.5.12.2 `enet_status_t ENET_DRV_Deinit ( enet_dev_if_t * enetIfPtr )`

Parameters

|                  |                             |
|------------------|-----------------------------|
| <i>enetIfPtr</i> | The ENET context structure. |
|------------------|-----------------------------|

Returns

The execution status.

### 14.5.12.3 `enet_status_t ENET_DRV_UpdateRxBuffDescrip ( enet_dev_if_t * enetIfPtr, bool isBuffUpdate )`

This function updates the used receive buffer descriptor ring to ensure that the used BDS is correctly used. It cleans the status region and sets the control region of the used receive buffer descriptor. If the `isBufferUpdate` flag is set, the data buffer in the buffer descriptor is updated.

Parameters

|                     |                              |
|---------------------|------------------------------|
| <i>enetIfPtr</i>    | The ENET context structure.  |
| <i>isBuffUpdate</i> | The data buffer update flag. |

Returns

The execution status.

### 14.5.12.4 `enet_status_t ENET_DRV_CleanupTxBuffDescrip ( enet_dev_if_t * enetIfPtr )`

First, store the transmit frame error statistic and PTP timestamp of the transmitted packets. Second, clean up the used transmit buffer descriptors. If the PTP 1588 feature is open, this interface captures the 1588 timestamp. It is called by the transmit interrupt handler.



### Parameters

|                  |                             |
|------------------|-----------------------------|
| <i>enetIfPtr</i> | The ENET context structure. |
|------------------|-----------------------------|

### Returns

The execution status.

#### 14.5.12.5 `volatile enet_bd_struct_t* ENET_DRV_IncrRxBuffDescripIndex ( enet_dev_if_t * enetIfPtr, volatile enet_bd_struct_t * curBd )`

### Parameters

|                  |                                        |
|------------------|----------------------------------------|
| <i>enetIfPtr</i> | The ENET context structure.            |
| <i>curBd</i>     | The current buffer descriptor pointer. |

### Returns

the increased received buffer descriptor.

#### 14.5.12.6 `volatile enet_bd_struct_t* ENET_DRV_IncrTxBuffDescripIndex ( enet_dev_if_t * enetIfPtr, volatile enet_bd_struct_t * curBd )`

### Parameters

|                  |                                        |
|------------------|----------------------------------------|
| <i>enetIfPtr</i> | The ENET context structure.            |
| <i>curBd</i>     | The current buffer descriptor pointer. |

### Returns

the increased transmit buffer descriptor.

#### 14.5.12.7 `bool ENET_DRV_RxErrorStats ( enet_dev_if_t * enetIfPtr, volatile enet_bd_struct_t * curBd )`

This interface gets the error statistics of the received frame. Because the error information is in the last BD of a frame, this interface should be called when processing the last BD of a frame.

## ENET Peripheral Driver

### Parameters

|                  |                                |
|------------------|--------------------------------|
| <i>enetIfPtr</i> | The ENET context structure.    |
| <i>curBd</i>     | The current buffer descriptor. |

### Returns

The frame error status.

- True if the frame has an error.
- False if the frame does not have an error.

#### 14.5.12.8 void ENET\_DRV\_TxErrorStats ( enet\_dev\_if\_t \* *enetIfPtr*, volatile enet\_bd\_struct\_t \* *curBd* )

This interface gets the error statistics of the transmit frame. Because the error information is in the last BD of a frame, this interface should be called when processing the last BD of a frame.

### Parameters

|                  |                                |
|------------------|--------------------------------|
| <i>enetIfPtr</i> | The ENET context structure.    |
| <i>curBd</i>     | The current buffer descriptor. |

#### 14.5.12.9 enet\_status\_t ENET\_DRV\_ReceiveData ( enet\_dev\_if\_t \* *enetIfPtr* )

### Parameters

|                  |                             |
|------------------|-----------------------------|
| <i>enetIfPtr</i> | The ENET context structure. |
|------------------|-----------------------------|

### Returns

The execution status.

#### 14.5.12.10 enet\_status\_t ENET\_DRV\_InstallNetIfCall ( enet\_dev\_if\_t \* *enetIfPtr*, enet\_netif\_callback\_t *function* )

## Parameters

|                  |                                                        |
|------------------|--------------------------------------------------------|
| <i>enetIfPtr</i> | The ENET context structure.                            |
| <i>function</i>  | The ENET TCP/IP stack net interface callback function. |

## Returns

The execution status.

**14.5.12.11 `enet_status_t ENET_DRV_SendData ( enet_dev_if_t * enetIfPtr, uint32_t dataLen, uint32_t bdNumUsed )`**

## Parameters

|                  |                                          |
|------------------|------------------------------------------|
| <i>enetIfPtr</i> | The ENET context structure.              |
| <i>dataLen</i>   | The frame data length to be transmitted. |
| <i>bdNumUsed</i> | The buffer descriptor need to be used.   |

## Returns

The execution status.

**14.5.12.12 `void ENET_DRV_RxIRQHandler ( uint32_t instance )`**

## Parameters

|                 |                           |
|-----------------|---------------------------|
| <i>instance</i> | The ENET instance number. |
|-----------------|---------------------------|

**14.5.12.13 `void ENET_DRV_TxIRQHandler ( uint32_t instance )`**

## Parameters

|                 |                           |
|-----------------|---------------------------|
| <i>instance</i> | The ENET instance number. |
|-----------------|---------------------------|

**14.5.12.14 `void ENET_DRV_CalculateCrc32 ( uint8_t * address, uint32_t * crcValue )`**

## ENET Peripheral Driver

### Parameters

|                 |                                              |
|-----------------|----------------------------------------------|
| <i>address</i>  | The ENET Mac hardware address.               |
| <i>crcValue</i> | The calculated CRC value of the Mac address. |

### 14.5.12.15 `enet_status_t ENET_DRV_AddMulticastGroup ( uint32_t instance, uint8_t * address, uint32_t * hash )`

### Parameters

|                 |                                                |
|-----------------|------------------------------------------------|
| <i>instance</i> | The ENET instance number.                      |
| <i>address</i>  | The multicast group address.                   |
| <i>hash</i>     | The hash value of the multicast group address. |

### Returns

The execution status.

### 14.5.12.16 `enet_status_t ENET_DRV_LeaveMulticastGroup ( uint32_t instance, uint8_t * address )`

### Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>instance</i> | The ENET instance number.    |
| <i>address</i>  | The multicast group address. |

### Returns

The execution status.

### 14.5.12.17 `void enet_mac_enqueue_buffer ( void ** queue, void * buffer )`

### Parameters

---

|               |                                          |
|---------------|------------------------------------------|
| <i>queue</i>  | The buffer queue address.                |
| <i>buffer</i> | The buffer will add to the buffer queue. |

**14.5.12.18 void\* enet\_mac\_dequeue\_buffer ( void \*\* *queue* )**

Parameters

|              |                           |
|--------------|---------------------------|
| <i>queue</i> | The buffer queue address. |
|--------------|---------------------------|

Returns

The buffer will be dequeued from the buffer queue.

**14.5.13 Variable Documentation**

**14.5.13.1 ENET\_Type\* const g\_enetBase[]**

**14.5.13.2 const IRQn\_Type g\_enetTxIrqld[]**



---

## Chapter 15

### External Watchdog Timer (EWM)

#### 15.1 Overview

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

#### Modules

- [EWM HAL driver](#)
- [EWM Peripheral Driver](#)

## EWM HAL driver

### 15.2 EWM HAL driver

#### 15.2.1 Overview

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

#### Data Structures

- struct [ewm\\_config\\_t](#)  
*Data structure for EWM initialize. [More...](#)*

#### Enumerations

- enum [ewm\\_status\\_t](#) {  
[kStatus\\_EWM\\_Success](#) = 0x0U,  
[kStatus\\_EWM\\_Fail](#) = 0x01,  
[kStatus\\_EWM\\_NotInitialized](#) = 0x2U,  
[kStatus\\_EWM\\_NullArgument](#) = 0x3U }  
*ewm status return codes.*

#### Functions

- static void [EWM\\_HAL\\_Enable](#) (EWM\_Type \*base)  
*Enable the EWM.*
- static void [EWM\\_HAL\\_Disable](#) (EWM\_Type \*base)  
*Enable the EWM.*
- static bool [EWM\\_HAL\\_IsEnable](#) (EWM\_Type \*base)  
*Checks whether the EWM is enabled.*
- static void [EWM\\_HAL\\_SetIntCmd](#) (EWM\_Type \*base, bool enable)  
*Enable/Disable EWM interrupt.*
- static void [EWM\\_HAL\\_SetCmpLowRegValue](#) (EWM\_Type \*base, uint8\_t minServiceCycles)  
*Set EWM compare low register value.*
- static void [EWM\\_HAL\\_SetCmpHighRegValue](#) (EWM\_Type \*base, uint8\_t maxServiceCycles)  
*Set EWM compare high register value.*
- static void [EWM\\_HAL\\_Refresh](#) (EWM\_Type \*base)  
*Service EWM.*
- void [EWM\\_HAL\\_SetConfig](#) (EWM\_Type \*base, const [ewm\\_config\\_t](#) \*ewmConfigPtr)  
*Config EWM control register.*
- void [EWM\\_HAL\\_Init](#) (EWM\_Type \*base)  
*Restores the EWM module to reset value.*



## 15.2.2 Data Structure Documentation

### 15.2.2.1 struct ewm\_config\_t

This structure is used when initializing the EWM.

#### Data Fields

- bool [ewmEnable](#)  
*Enable EWM module.*
- bool [ewmInEnable](#)  
*Enable EWM\_in input enable.*
- bool [ewmInAssertLogic](#)  
*Set EWM\_in signal assertion state.*
- bool [intEnable](#)  
*Enable EWM interrupt enable.*
- uint8\_t [ewmCmpLowValue](#)  
*Set EWM compare low register value.*
- uint8\_t [ewmCmpHighValue](#)  
*Set EWM compare high register value, the maximum value should be 0xfe otherwise the counter will never expire.*

## 15.2.3 Enumeration Type Documentation

### 15.2.3.1 enum ewm\_status\_t

Enumerator

- kStatus\_EWM\_Success* EWM operation Succeed.
- kStatus\_EWM\_Fail* EWM operation Failed.
- kStatus\_EWM\_NotInitialized* EWM is not initialized yet.
- kStatus\_EWM\_NullArgument* Argument is NULL.

## 15.2.4 Function Documentation

### 15.2.4.1 static void EWM\_HAL\_Enable ( EWM\_Type \* base ) [inline], [static]

This function checks whether the EWM is enabled.

Parameters

---

## EWM HAL driver

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The EWM peripheral base address |
|-------------|---------------------------------|

### 15.2.4.2 static void EWM\_HAL\_Disable ( EWM\_Type \* *base* ) [inline], [static]

This function checks whether the EWM is enabled.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The EWM peripheral base address |
|-------------|---------------------------------|

### 15.2.4.3 static bool EWM\_HAL\_IsEnable ( EWM\_Type \* *base* ) [inline], [static]

This function checks whether the EWM is enabled.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The EWM peripheral base address |
|-------------|---------------------------------|

Returns

State of the module

Return values

|              |                       |
|--------------|-----------------------|
| <i>false</i> | means EWM is disabled |
| <i>true</i>  | means WODG is enabled |

### 15.2.4.4 static void EWM\_HAL\_SetIntCmd ( EWM\_Type \* *base*, bool *enable* ) [inline], [static]

This function sets EWM enable/disable.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | The EWM peripheral base address  |
| <i>enable</i> | Set EWM interrupt enable/disable |

**15.2.4.5 static void EWM\_HAL\_SetCmpLowRegValue ( EWM\_Type \* *base*, 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.

Parameters

|                         |                                    |
|-------------------------|------------------------------------|
| <i>base</i>             | The EWM peripheral base address    |
| <i>minServiceCycles</i> | The EWM compare low register value |

**15.2.4.6 static void EWM\_HAL\_SetCmpHighRegValue ( EWM\_Type \* *base*, 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>base</i>             | The EWM peripheral base address    |
| <i>maxServiceCycles</i> | The EWM compare low register value |

**15.2.4.7 static void EWM\_HAL\_Refresh ( EWM\_Type \* *base* ) [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>base</i> | The EWM peripheral base address |
|-------------|---------------------------------|

**15.2.4.8 void EWM\_HAL\_SetConfig ( EWM\_Type \* *base*, const ewm\_config\_t \* *ewmConfigPtr* )**

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.

## EWM HAL driver

Parameters

|                     |                                 |
|---------------------|---------------------------------|
| <i>base</i>         | The EWM peripheral base address |
| <i>ewmConfigPtr</i> | config EWM CTRL register        |

### 15.2.4.9 void EWM\_HAL\_Init ( EWM\_Type \* *base* )

This function restores the EWM module to reset value.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The EWM peripheral base address |
|-------------|---------------------------------|

## 15.3 EWM Peripheral Driver

### 15.3.1 Overview

The section describes the programming interface of the EWM Peripheral driver. The EWM driver provides an easy way to initialize and configure the EWM.

### 15.3.2 EWM Initialization

To initialize the EWM module, call the [EWM\\_DRV\\_Init\(\)](#) function and pass in the user configuration structure. This function automatically enables the EWM module and clock. After the [EWM\\_DRV\\_Init\(\)](#) function is called, the EWM is enabled and its counter is working. Therefore, the [EWM\\_DRV\\_Refresh\(\)](#) function should be called before the EWM times out.

This example code shows how to initialize and configure the driver:

```
// Define device configuration.
const ewm_user_config_t init =
{
 .ewmEnable = true, // When reset EWM_out signal is assert, to de-assert the signal EWM must be enabled
 //
 .ewmInAssertionState = kEWMLogicOneAssert, // At reset EWM_in is assert in logic zero, so EWM_in signal
 assertion should be set in logic one, otherwise,
 EWM_out will be assert when servicing EWM counter //
 .ewmInputEnable = true, // Enable the EWM_in signal, so EWM can monitor external circuit //
 .ewmIntEnable = true, // Enable EWM interrupt //
 .ewmCmpLowValue = 0x00, // The compare low register configures the minimum period to service the EWM
 counter //
 .ewmCmpHighValue = 0xfe, // 0xfe is the maximum value for compare high register, because interrupt
 occurs when counter value is greater than compare high register value //
};
```

Initialize EWM. `EWM_DRV_Init(instance, &init);`

### 15.3.3 EWM Refresh

After the EWM is enabled, the [EWM\\_DRV\\_Refresh\(\)](#) function should be called periodically to prevent the EWM from timing out. Otherwise, `EWM_out` is asserted and an EWM interrupt occurs. When `EWM_in` is asserted, servicing EWM causes the `EWM_out` signal to be asserted.

#### Variables

- `EWM_Type *const g_ewmBase []`  
*Table of base addresses for EWM instances.*
- `const IRQn_Type g_ewmIrqld [EWM_INSTANCE_COUNT]`  
*Table to save EWM IRQ enumeration numbers defined in the CMSIS header file.*

## EWM Peripheral Driver

### EWM Driver

- `ewm_status_t EWM_DRV_Init` (uint32\_t instance, const `ewm_config_t *ConfigPtr`)  
*Initializes the EWM.*
- void `EWM_DRV_Deinit` (uint32\_t instance)  
*Closes the clock for EWM.*
- void `EWM_DRV_Refresh` (uint32\_t instance)  
*Refreshes the EWM.*
- bool `EWM_DRV_IsRunning` (uint32\_t instance)  
*Gets the EWM running status.*
- void `EWM_DRV_SetIntCmd` (uint32\_t instance, bool enable)  
*Enables/disables the EWM interrupt.*

### 15.3.4 Function Documentation

#### 15.3.4.1 `ewm_status_t EWM_DRV_Init` ( uint32\_t *instance*, const `ewm_config_t *ConfigPtr` )

This function initializes the EWM. When called, the EWM runs according to the configuration.

Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>instance</i>  | EWM instance ID                                           |
| <i>ConfigPtr</i> | EWM user configure data structure, see #EWM_user_config_t |

Returns

Execution status.

#### 15.3.4.2 void `EWM_DRV_Deinit` ( uint32\_t *instance* )

This function sets the run time array to zero and closes the clock.

Parameters

|                 |                 |
|-----------------|-----------------|
| <i>instance</i> | EWM instance ID |
|-----------------|-----------------|

#### 15.3.4.3 void `EWM_DRV_Refresh` ( uint32\_t *instance* )

This function feeds the EWM. It sets the EWM timer count to zero and should be called before the EWM timer times out.

Parameters

|                 |                 |
|-----------------|-----------------|
| <i>instance</i> | EWM instance ID |
|-----------------|-----------------|

#### 15.3.4.4 bool EWM\_DRV\_IsRunning ( uint32\_t *instance* )

This function gets the EWM running status.

Parameters

|                 |                 |
|-----------------|-----------------|
| <i>instance</i> | EWM instance ID |
|-----------------|-----------------|

Returns

EWM running status. False means not running. True means running

#### 15.3.4.5 void EWM\_DRV\_SetIntCmd ( uint32\_t *instance*, bool *enable* )

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>instance</i> | EWM instance ID.              |
| <i>enable</i>   | EWM interrupt enable/disable. |

### 15.3.5 Variable Documentation

#### 15.3.5.1 EWM\_Type\* const g\_ewmBase[]

#### 15.3.5.2 const IRQn\_Type g\_ewmIrqId[EWM\_INSTANCE\_COUNT]





## Chapter 16

# C90TFS Flash Driver

### 16.1 Overview

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

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. Therefore, 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 allows the user to use this driver in the background debug mode without returning to the caller 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. To use callback in the application, this callback function must not be placed in the same Flash block in which a program/erase operation occurs to avoid the RWW error.
3. To suspend the sector erase operation for a simple method, invoke the FlashEraseSuspend function within callback of FlashEraseSector. In this case, the FlashEraseSuspend must not be placed in the same block in which the 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 the 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 standard demo included in the release package provides the code part to disable/enable the Flash cache.

### Data Structures

- struct [PFLASH\\_SSD\\_CONFIG](#)  
*Flash SSD Configuration Structure. [More...](#)*

### Macros

- #define [SET\\_FLASH\\_INT\\_BITS](#)(ftfxRegBase, value)

## Overview

- Sets the Flash interrupt enable bits in the FCNFG register.
- #define `GET_FLASH_INT_BITS(ftfxRegBase)`  
Returns 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_FLEX_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)*



## Overview

- *FlexNVM partition code 1010 mapping to data flash size in bytes (0xFFFFFFFF = reserved)*  
 • #define **DEPART\_1011** FSL\_FEATURE\_FLASH\_FLEX\_NVM\_DFLASH\_SIZE\_FOR\_DEPART\_1011
- *FlexNVM partition code 1011 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 1100 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 1101 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 1110 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 1111 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)  
*Relocates a function to RAM address.*
- uint32\_t [FlashInit](#) (PFLASH\_SSD\_CONFIG pSSDConfig)  
*Initializes Flash.*
- 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)  
*Calculates 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  
*Cannot 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 pointer.*
- typedef uint32\_t(\* **PFLASHINIT** )(PFLASH\_SSD\_CONFIG pSSDConfig)  
*FlashInit function pointer.*
- typedef uint32\_t(\* **PFLASHGETPROTECTION** )(PFLASH\_SSD\_CONFIG pSSDConfig, uint32\_t \*protectStatus)  
*PFlashGetProtection function pointer.*
- typedef uint32\_t(\* **PFLASHSETPROTECTION** )(PFLASH\_SSD\_CONFIG pSSDConfig, uint32\_t protectStatus)  
*PFlashSetProtection function pointer.*
- typedef uint32\_t(\* **PFLASHGETSECURITYSTATE** )(PFLASH\_SSD\_CONFIG pSSDConfig, uint8\_t \*securityState)  
*FlashGetSecurityState function pointer.*
- typedef uint32\_t(\* **PFLASHSECURITYBYPASS** )(PFLASH\_SSD\_CONFIG pSSDConfig, uint8\_t \*keyBuffer, **PFLASHCOMMANDSEQUENCE** pFlashCommandSequence)  
*FlashSecurityByPass function pointer.*
- typedef uint32\_t(\* **PFLASHERASEALLBLOCK** )(PFLASH\_SSD\_CONFIG pSSDConfig, **PFLASHCOMMANDSEQUENCE** pFlashCommandSequence)  
*FlashEraseAllBlock function pointer.*
- typedef uint32\_t(\* **PFLASHERASEBLOCK** )(PFLASH\_SSD\_CONFIG pSSDConfig, uint32\_t dest, **PFLASHCOMMANDSEQUENCE** pFlashCommandSequence)  
*FlashEraseBlock function pointer.*
- typedef uint32\_t(\* **PFLASHERASESECTOR** )(PFLASH\_SSD\_CONFIG pSSDConfig, uint32\_t dest, uint32\_t size, **PFLASHCOMMANDSEQUENCE** pFlashCommandSequence)  
*FlashEraseSector function pointer.*
- typedef uint32\_t(\* **PFLASHERASESUSPEND** )(PFLASH\_SSD\_CONFIG pSSDConfig)  
*FlashEraseSuspend function pointer.*
- typedef uint32\_t(\* **PFLASHERASERESUME** )(PFLASH\_SSD\_CONFIG pSSDConfig)  
*FlashEraseResume function pointer.*
- typedef uint32\_t(\* **PFLASHPROGRAMSECTION** )(PFLASH\_SSD\_CONFIG pSSDConfig, uint32\_t dest, uint16\_t number, **PFLASHCOMMANDSEQUENCE** pFlashCommandSequence)  
*FlashProgramSection function pointer.*
- typedef uint32\_t(\* **PFLASHCHECKSUM** )(PFLASH\_SSD\_CONFIG pSSDConfig, uint32\_t dest, uint32\_t size, uint32\_t \*pSum)  
*FlashChecksum function pointer.*

## Overview

- typedef uint32\_t(\* [pFLASHVERIFYALLBLOCK](#) )(PFLASH\_SSD\_CONFIG pSSDConfig, uint8\_t marginLevel, [pFLASHCOMMANDSEQUENCE](#) pFlashCommandSequence)  
*FlashVerifyAllBlock function pointer.*
- 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 pointer.*
- typedef uint32\_t(\* [pFLASHREADONCE](#) )(PFLASH\_SSD\_CONFIG pSSDConfig, uint8\_t \*pDataArray, [pFLASHCOMMANDSEQUENCE](#) pFlashCommandSequence)  
*FlashReadOnce function pointer.*
- typedef uint32\_t(\* [pFLASHPROGRAMONCE](#) )(PFLASH\_SSD\_CONFIG pSSDConfig, uint8\_t \*pDataArray, [pFLASHCOMMANDSEQUENCE](#) pFlashCommandSequence)  
*FlashProgramOnce function pointer.*
- 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 pointer.*
- typedef uint32\_t(\* [pFLASHREADRESOURCE](#) )(PFLASH\_SSD\_CONFIG pSSDConfig, uint32\_t dest, uint8\_t \*pDataArray, uint8\_t resourceSelectCode, [pFLASHCOMMANDSEQUENCE](#) pFlashCommandSequence)  
*FlashReadResource function pointer.*
- typedef uint32\_t(\* [pFLASHPROGRAM](#) )(PFLASH\_SSD\_CONFIG pSSDConfig, uint32\_t dest, uint32\_t size, uint8\_t \*pData, [pFLASHCOMMANDSEQUENCE](#) pFlashCommandSequence)  
*FlashProgram function pointer.*
- 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 pointer.*
- typedef uint32\_t(\* [pFLASHSWAP](#) )(PFLASH\_SSD\_CONFIG pSSDConfig, uint32\_t flashAddress, [PFLASH\\_SWAP\\_CALLBACK](#) pSwapCallback, [pFLASHCOMMANDSEQUENCE](#) pFlashCommandSequence)  
*PFlashSwap function pointer.*
- typedef uint32\_t(\* [pDFLASHGETPROTECTION](#) )(PFLASH\_SSD\_CONFIG pSSDConfig, uint8\_t \*protectStatus)  
*DFlashGetProtection function pointer.*
- typedef uint32\_t(\* [pDFLASHSETPROTECTION](#) )(PFLASH\_SSD\_CONFIG pSSDConfig, uint8\_t protectStatus)  
*DFlashSetProtection function pointer.*
- typedef uint32\_t(\* [pEERAMGETPROTECTION](#) )(PFLASH\_SSD\_CONFIG pSSDConfig, uint8\_t \*protectStatus)  
*EERAMGetProtection function pointer.*
- typedef uint32\_t(\* [pEERAMSETPROTECTION](#) )(PFLASH\_SSD\_CONFIG pSSDConfig, uint8\_t protectStatus)  
*EERAMSetProtection function pointer.*
- typedef uint32\_t(\* [pDEFLASHPARTITION](#) )(PFLASH\_SSD\_CONFIG pSSDConfig, uint8\_t E-



EEDataSizeCode, uint8\_t DEPartitionCode, [pFLASHCOMMANDSEQUENCE](#) pFlashCommandSequence)

*DEFlashPartition function pointer.*

- typedef uint32\_t(\* [pSETEEEENABLE](#) )(PFLASH\_SSD\_CONFIG pSSDConfig, uint8\_t EEEnable, [pFLASHCOMMANDSEQUENCE](#) pFlashCommandSequence)

*SetEEEnable function pointer.*

- typedef uint32\_t(\* [pEEEWWRITE](#) )(PFLASH\_SSD\_CONFIG pSSDConfig, uint32\_t dest, uint32\_t size, uint8\_t \*pData)

*EEEWrite function pointer.*

## 16.2 Data Structure Documentation

### 16.2.1 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 ftxRegBase, PFlashBlockBase, PFlashBlockSize, DFlashBlockBase, EERAMBlockBase, DebugEnable and CallBack before passing the structure to SSD functions. The rest of parameters such as DFlashBlockSize, and EEBlockSize 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 [ftxRegBase](#)  
*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

### 16.3 Macro Definition Documentation

#### 16.3.1 #define BYTE2WORD( x ) (x)

Two address types are only different in DSC devices. In Kinstis devices, they are the same

#### 16.3.2 #define WORD2BYTE( x ) (x)

Two address types are only different in DSC devices. In Kinstis devices, they are the same

#### 16.3.3 #define SET\_FLASH\_INT\_BITS( *ftfxRegBase*, *value* )

##### Value:

```
REG_WRITE((ftfxRegBase) + FTFx_SSD_FCENFG_OFFSET, \
 ((value)&(FTFx_SSD_FCENFG_CCIE | FTFx_SSD_FCENFG_RDCOLLIE)))
```

##### Parameters

|                       |                                                                                                                                                                                                                                             |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>ftfxRegBase</i> ,: | Specifies register base address of the 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. |

#### 16.3.4 #define GET\_FLASH\_INT\_BITS( *ftfxRegBase* )

##### Value:

```
REG_READ((ftfxRegBase) + FTFx_SSD_FCENFG_OFFSET) &\
 (FTFx_SSD_FCENFG_CCIE | FTFx_SSD_FCENFG_RDCOLLIE)
```

##### Parameters

|                       |                                                      |
|-----------------------|------------------------------------------------------|
| <i>ftfxRegBase</i> ,: | Specifies register base address of the Flash module. |
|-----------------------|------------------------------------------------------|

#### 16.3.5 #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

### 16.3.6 #define FTFx\_ERR\_PVIOL 0x0010U

Possible causes:

FPVIOL 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.

### 16.3.7 #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.

### 16.3.8 #define FTFx\_ERR\_CHANGEPROT 0x0100U

Possible causes:

Violates protection transition.

Solution:

In NVM normal mode, protection size cannot be decreased. Therefore, the only increasing protection size is permitted if the device is operating in this mode.

### 16.3.9 #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

### 16.3.10 #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.

### 16.3.11 #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.

### 16.3.12 #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.

### 16.3.13 #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.

## 16.4 Function Documentation

### 16.4.1 uint32\_t RelocateFunction ( uint32\_t *dest*, uint32\_t *size*, uint32\_t *src* )

This function provides a facility to relocate a function 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 .

### 16.4.2 uint32\_t FlashInit ( PFLASH\_SSD\_CONFIG pSSDConfig )

This API initializes 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)

### 16.4.3 uint32\_t FlashCommandSequence ( PFLASH\_SSD\_CONFIG pSSDConfig )

This API is used to perform command write sequence on 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\_M-GSTAT0)

## Function Documentation

### 16.4.4 uint32\_t PFlashGetProtection ( PFLASH\_SSD\_CONFIG pSSDConfig, uint32\_t \* protectStatus )

This API retrieves the 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"> <li>• 0: this area is protected.</li> <li>• 1: this area is unprotected.</li> </ul> |

#### Returns

Successful completion (FTFx\_OK)

### 16.4.5 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 a setting violation, it returns 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 shown below: <ul style="list-style-type: none"> <li>• 0: this area is protected.</li> <li>• 1: this area is unprotected.</li> </ul> |

**Returns**

- Successful completion (FTFx\_OK )
- Error value (FTFx\_ERR\_CHANGEPROT)

**16.4.6 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)

**16.4.7 uint32\_t FlashSecurityBypass ( PFLASH\_SSD\_CONFIG pSSDConfig, uint8\_t \* keyBuffer, pFLASHCOMMANDSEQUENCE pFlashCommandSequence )**

This API un-secures the device by comparing the user’s provided back door key with the ones in the Flash Configuration Field. If they are matched, the security is released. Otherwise, an error code is 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

### 16.4.8 `uint32_t FlashEraseAllBlock ( PFLASH_SSD_CONFIG pSSDConfig, pFLASHCOMMANDSEQUENCE pFlashCommandSequence )`

This API erases all Flash memory, initializes the FlexRAM, verifies all memory contents, and then releases the 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)

**16.4.9 uint32\_t FlashVerifyAllBlock ( PFLASH\_SSD\_CONFIG *pSSDConfig*, uint8\_t *marginLevel*, pFLASHCOMMANDSEQUENCE *pFlashCommandSequence* )**

This function checks 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 releases security if the readout passes.

Parameters

|                                |                                                                                                                                                                                  |
|--------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>pSSDConfig</i> ,:           | The SSD configuration structure pointer.                                                                                                                                         |
| <i>marginLevel</i> ,:          | Read Margin Choice as follows: <i>marginLevel</i> = 0x0: use the Normal read level<br><i>marginLevel</i> = 0x1: use the User read <i>marginLevel</i> = 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)

**16.4.10 uint32\_t FlashEraseSector ( PFLASH\_SSD\_CONFIG *pSSDConfig*, uint32\_t *dest*, uint32\_t *size*, pFLASHCOMMANDSEQUENCE *pFlashCommandSequence* )**

This API erases one or more sectors in P-Flash or D-Flash memory. This API always returns FTFx\_OK if size provided by the 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\_SIZE)

### 16.4.11 uint32\_t FlashVerifySection ( PFLASH\_SSD\_CONFIG pSSD-Config, uint32\_t dest, uint16\_t number, uint8\_t marginLevel, PFLASHCOMMANDSEQUENCE pFlashCommandSequence )

This API checks if a part of the P-Flash or the 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)

#### 16.4.12 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 the RWW error.

## Function Documentation

### Parameters

|                      |                                          |
|----------------------|------------------------------------------|
| <i>pSSDConfig</i> ,: | The SSD configuration structure pointer. |
|----------------------|------------------------------------------|

### Returns

Successful completion (FTFx\_OK)

### 16.4.13 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)

### 16.4.14 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. See the corresponding reference manual to get the 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)

**16.4.15 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. See the 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)

**16.4.16 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>resourceSelect-Code</i> ,:  | Read resource select code: 0 : Flash IFR 1: Version ID     |
| <i>pFlash-Command-Sequence</i> | : Pointer to the Flash command sequence function.          |

### Returns

Successful completion (FTFx\_OK)  
 Error value (FTFx\_ERR\_ACCERR)

### 16.4.17 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>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\_SIZE, FTFx\_ERR\_MGSTAT0)

### 16.4.18 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)

### 16.4.19 uint32\_t FlashChecksum ( PFLASH\_SSD\_CONFIG pSSDConfig, uint32\_t dest, uint32\_t size, uint32\_t \* pSum )

This API performs 32 bit sum of each byte data over a 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 the 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)

### 16.4.20 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 FlashProgram-Section, 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-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, FTFx\_ERR\_R-AMRDY)



### 16.4.21 `uint32_t FlashEraseBlock ( PFLASH_SSD_CONFIG pSSDConfig, uint32_t dest, pFLASHCOMMANDSEQUENCE pFlashCommandSequence )`

This API erases 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 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)

### 16.4.22 uint32\_t FlashVerifyBlock ( PFLASH\_SSD\_CONFIG pSSDConfig, uint32\_t dest, uint8\_t marginLevel, pFLASHCOMMANDSEQUENCE pFlashCommandSequence )

This API checks 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 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 17

# External Bus Interface (FLEXBUS)

### 17.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the External Bus Interface (FLEXBUS) block of Kinetis devices.

#### Modules

- [FLEXBUS HAL driver](#)
- [FLEXBUS Peripheral Driver](#)

## FLEXBUS HAL driver

### 17.2 FLEXBUS HAL driver

#### 17.2.1 Overview

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

#### Files

- file [fsl\\_flexbus\\_hal.h](#)

#### Data Structures

- struct [flexbus\\_user\\_config\\_t](#)  
*Configuration structure that the user needs to set. [More...](#)*

#### Enumerations

- enum [flexbus\\_status\\_t](#)  
*Flexbus status return codes.*
- enum [flexbus\\_port\\_size\\_t](#) {  
[kFlexbus4bytes](#) = 0x00U,  
[kFlexbus1byte](#) = 0x01U,  
[kFlexbus2bytes](#) = 0x02U }  
*Defines port size for Flexbus peripheral.*
- enum [flexbus\\_write\\_address\\_hold\\_t](#) {  
[kFlexbusHold1cycle](#) = 0x00U,  
[kFlexbusHold2cycles](#) = 0x01U,  
[kFlexbusHold3cycles](#) = 0x02U,  
[kFlexbusHold4cycles](#) = 0x03U }  
*Defines number of cycles to hold address and attributes for Flexbus peripheral.*
- enum [flexbus\\_read\\_address\\_hold\\_t](#) {  
[kFlexbusHold4or3cycles](#) = 0x03U,  
[kFlexbusHold3or2cycles](#) = 0x02U,  
[kFlexbusHold2or1cycle](#) = 0x01U,  
[kFlexbusHold1or0cycle](#) = 0x00U }  
*Defines number of cycles to hold address and attributes for Flexbus peripheral.*
- enum [flexbus\\_address\\_setup\\_t](#) {  
[kFlexbusFirstRisingEdge](#) = 0x00U,  
[kFlexbusSecondRisingEdge](#) = 0x01U,  
[kFlexbusThirdRisingEdge](#) = 0x02U,  
[kFlexbusFourthRisingEdge](#) = 0x03U }  
*Address setup for Flexbus peripheral.*
- enum [flexbus\\_bytelane\\_shift\\_t](#) {  
[kFlexbusNotShifted](#) = 0x00U,  
[kFlexbusShifted](#) = 0x01U }

- Defines byte-lane shift for Flexbus peripheral.*

  - enum `flexbus_multiplex_group1_t` {  
`kFlexbusMultiplexGroup1_FB_ALE` = 0x00U,  
`kFlexbusMultiplexGroup1_FB_CS1` = 0x01U,  
`kFlexbusMultiplexGroup1_FB_TS` = 0x02U }

*Defines multiplex group1 valid signals.*

  - enum `flexbus_multiplex_group2_t` {  
`kFlexbusMultiplexGroup2_FB_CS4` = 0x00U,  
`kFlexbusMultiplexGroup2_FB_TSIZ0` = 0x01U,  
`kFlexbusMultiplexGroup2_FB_BE_31_24` = 0x02U }

*Defines multiplex group2 valid signals.*

  - enum `flexbus_multiplex_group3_t` {  
`kFlexbusMultiplexGroup3_FB_CS5` = 0x00U,  
`kFlexbusMultiplexGroup3_FB_TSIZ1` = 0x01U,  
`kFlexbusMultiplexGroup3_FB_BE_23_16` = 0x02U }

*Defines multiplex group3 valid signals.*

  - enum `flexbus_multiplex_group4_t` {  
`kFlexbusMultiplexGroup4_FB_TBST` = 0x00U,  
`kFlexbusMultiplexGroup4_FB_CS2` = 0x01U,  
`kFlexbusMultiplexGroup4_FB_BE_15_8` = 0x02U }

*Defines multiplex group4 valid signals.*

  - enum `flexbus_multiplex_group5_t` {  
`kFlexbusMultiplexGroup5_FB_TA` = 0x00U,  
`kFlexbusMultiplexGroup5_FB_CS3` = 0x01U,  
`kFlexbusMultiplexGroup5_FB_BE_7_0` = 0x02U }

*Defines multiplex group5 valid signals.*

## Configuration

- void `FLEXBUS_HAL_Init` (FB\_Type \*base)  
*Initialization to default values.*
- void `FLEXBUS_HAL_Configure` (FB\_Type \*base, const `flexbus_user_config_t` \*userConfigPtr)  
*Configure to a known values.*
- static void `FLEXBUS_HAL_WriteAddr` (FB\_Type \*base, uint8\_t chip, uint16\_t addr, uint16\_t addrMask)  
*Write chip-select base address.*
- static void `FLEXBUS_HAL_SetChipSelectValidCmd` (FB\_Type \*base, uint8\_t chip, bool valid)  
*Sets chip-selects valid bit or not.*
- static void `FLEXBUS_HAL_SetWriteProtectionCmd` (FB\_Type \*base, uint8\_t chip, bool enable)  
*Enables or disables write protection function for Flexbus.*
- static void `FLEXBUS_HAL_SetBurstWriteCmd` (FB\_Type \*base, uint8\_t chip, bool enable)  
*Enables or disables burst-write on Flexbus.*
- static void `FLEXBUS_HAL_SetBurstReadCmd` (FB\_Type \*base, uint8\_t chip, bool enable)  
*Enables or disables burst-read bit on Flexbus.*
- static void `FLEXBUS_HAL_SetByteModeCmd` (FB\_Type \*base, uint8\_t chip, bool enable)  
*Enables or disables byte-enable support on Flexbus.*
- static void `FLEXBUS_HAL_SetPortSize` (FB\_Type \*base, uint8\_t chip, `flexbus_port_size_t` size)  
*Sets port size on Flexbus.*

## FLEXBUS HAL driver

- static void [FLEXBUS\\_HAL\\_SetAutoAcknowledgeCmd](#) (FB\_Type \*base, uint8\_t chip, bool enable)  
*Enables auto-acknowledge on Flexbus.*
- static void [FLEXBUS\\_HAL\\_SetByteLaneShift](#) (FB\_Type \*base, uint8\_t chip, flexbus\_bytelane\_shift\_t shift)  
*Enables byte-lane shift on Flexbus.*
- static void [FLEXBUS\\_HAL\\_SetWaitStates](#) (FB\_Type \*base, uint8\_t chip, uint8\_t waitStates)  
*Sets number of wait states on Flexbus.*
- static void [FLEXBUS\\_HAL\\_SetWriteAddrHoldOrDeselect](#) (FB\_Type \*base, uint8\_t chip, flexbus\_write\_address\_hold\_t addrHold)  
*Sets write address hold or deselect.*
- static void [FLEXBUS\\_HAL\\_SetReadAddrHoldOrDeselect](#) (FB\_Type \*base, uint8\_t chip, flexbus\_read\_address\_hold\_t addrHold)  
*Sets read address hold or deselect.*
- static void [FLEXBUS\\_HAL\\_SetAddrSetup](#) (FB\_Type \*base, uint8\_t chip, flexbus\_address\_setup\_t delay)  
*Set address setup.*
- static void [FLEXBUS\\_HAL\\_SetExtendedAddrLatchCmd](#) (FB\_Type \*base, uint8\_t chip, bool enable)  
*Enables extended address latch.*
- static void [FLEXBUS\\_HAL\\_SetSecondaryWaitStateCmd](#) (FB\_Type \*base, uint8\_t chip, bool enable)  
*Enables secondary wait state.*
- static void [FLEXBUS\\_HAL\\_SetMultiplexControlGroup1](#) (FB\_Type \*base, flexbus\_multiplex\_group1\_t controls)  
*Multiplex group1 set.*
- static flexbus\_multiplex\_group1\_t [FLEXBUS\\_HAL\\_GetMultiplexControlGroup1](#) (FB\_Type \*base)  
*Multiplex group1 get.*
- static void [FLEXBUS\\_HAL\\_SetMultiplexControlGroup2](#) (FB\_Type \*base, flexbus\_multiplex\_group2\_t controls)  
*Multiplex group2 set.*
- static flexbus\_multiplex\_group2\_t [FLEXBUS\\_HAL\\_GetMultiplexControlGroup2](#) (FB\_Type \*base)  
*Multiplex group2 get.*
- static void [FLEXBUS\\_HAL\\_SetMultiplexControlGroup3](#) (FB\_Type \*base, flexbus\_multiplex\_group3\_t controls)  
*Multiplex group3 set.*
- static flexbus\_multiplex\_group3\_t [FLEXBUS\\_HAL\\_GetMultiplexControlGroup3](#) (FB\_Type \*base)  
*Multiplex group3 get.*
- static void [FLEXBUS\\_HAL\\_SetMultiplexControlGroup4](#) (FB\_Type \*base, flexbus\_multiplex\_group4\_t controls)  
*Multiplex group4 set.*
- static flexbus\_multiplex\_group4\_t [FLEXBUS\\_HAL\\_GetMultiplexControlGroup4](#) (FB\_Type \*base)  
*Multiplex group4 get.*
- static void [FLEXBUS\\_HAL\\_SetMultiplexControlGroup5](#) (FB\_Type \*base, flexbus\_multiplex\_group5\_t controls)

- Multiplex group5 set.*

  - static `flexbus_multiplex_group5_t` `FLEXBUS_HAL_GetMultiplexControlGroup5` (FB\_Type \*base)

*Multiplex group5 get.*

## 17.2.2 Data Structure Documentation

### 17.2.2.1 struct flexbus\_user\_config\_t

#### Data Fields

- uint8\_t `chip`  
*Chip FlexBus for validation.*
- uint8\_t `waitStates`  
*Value of wait states.*
- uint32\_t `baseAddress`  
*Base address for using FlexBus.*
- uint32\_t `baseAddressMask`  
*Base address mask.*
- bool `writeProtect`  
*Write protected.*
- bool `burstWrite`  
*Burst-Write enable.*
- bool `burstRead`  
*Burst-Read enable.*
- bool `byteEnableMode`  
*Byte-enable mode support.*
- bool `autoAcknowledge`  
*Auto acknowledge setting.*
- bool `extendTransferAddress`  
*Extend transfer start/extend address latch enable.*
- bool `secondaryWaitStates`  
*Secondary wait states number.*
- flexbus\_port\_size\_t `portSize`  
*Port size of transfer.*
- flexbus\_bytelane\_shift\_t `byteLaneShift`  
*Byte-lane shift enable.*
- flexbus\_write\_address\_hold\_t `writeAddressHold`  
*Write address hold or deselect option.*
- flexbus\_read\_address\_hold\_t `readAddressHold`  
*Read address hold or deselect option.*
- flexbus\_address\_setup\_t `addressSetup`  
*Address setup setting.*
- flexbus\_multiplex\_group1\_t `group1MultiplexControl`  
*FlexBus Signal Group 1 Multiplex control.*
- flexbus\_multiplex\_group2\_t `group2MultiplexControl`  
*FlexBus Signal Group 2 Multiplex control.*
- flexbus\_multiplex\_group3\_t `group3MultiplexControl`  
*FlexBus Signal Group 3 Multiplex control.*

## FLEXBUS HAL driver

- [flexbus\\_multiplex\\_group4\\_t group4MultiplexControl](#)  
*FlexBus Signal Group 4 Multiplex control.*
- [flexbus\\_multiplex\\_group5\\_t group5MultiplexControl](#)  
*FlexBus Signal Group 5 Multiplex control.*

### 17.2.3 Enumeration Type Documentation

#### 17.2.3.1 enum flexbus\_status\_t

#### 17.2.3.2 enum flexbus\_port\_size\_t

Enumerator

- kFlexbus4bytes* 32-bit port size
- kFlexbus1byte* 8-bit port size
- kFlexbus2bytes* 16-bit port size

#### 17.2.3.3 enum flexbus\_write\_address\_hold\_t

Enumerator

- kFlexbusHold1cycle* Hold address and attributes one cycle after FB\_CS<sub>n</sub> negates on writes.
- kFlexbusHold2cycles* Hold address and attributes two cycle after FB\_CS<sub>n</sub> negates on writes.
- kFlexbusHold3cycles* Hold address and attributes three cycle after FB\_CS<sub>n</sub> negates on writes.
- kFlexbusHold4cycles* Hold address and attributes four cycle after FB\_CS<sub>n</sub> negates on writes.

#### 17.2.3.4 enum flexbus\_read\_address\_hold\_t

Enumerator

- kFlexbusHold4or3cycles* Hold address and attributes 4 or 3 cycles on reads.
- kFlexbusHold3or2cycles* Hold address and attributes 3 or 2 cycles on reads.
- kFlexbusHold2or1cycle* Hold address and attributes 2 or 1 cycles on reads.
- kFlexbusHold1or0cycle* Hold address and attributes 1 or 0 cycles on reads.

#### 17.2.3.5 enum flexbus\_address\_setup\_t

Enumerator

- kFlexbusFirstRisingEdge* Assert FB\_CS<sub>n</sub> on first rising clock edge after address is asserted.
- kFlexbusSecondRisingEdge* Assert FB\_CS<sub>n</sub> on second rising clock edge after address is asserted.
- kFlexbusThirdRisingEdge* Assert FB\_CS<sub>n</sub> on third rising clock edge after address is asserted.
- kFlexbusFourthRisingEdge* Assert FB\_CS<sub>n</sub> on fourth rising clock edge after address is asserted.



### 17.2.3.6 enum flexbus\_bytelane\_shift\_t

Enumerator

*kFlexbusNotShifted* Not shifted. Data is left-justified on FB\_AD.

*kFlexbusShifted* Shifted. Data is right justified on FB\_AD.

### 17.2.3.7 enum flexbus\_multiplex\_group1\_t

Enumerator

*kFlexbusMultiplexGroup1\_FB\_ALE* FB\_ALE.

*kFlexbusMultiplexGroup1\_FB\_CS1* FB\_CS1.

*kFlexbusMultiplexGroup1\_FB\_TS* FB\_TS.

### 17.2.3.8 enum flexbus\_multiplex\_group2\_t

Enumerator

*kFlexbusMultiplexGroup2\_FB\_CS4* FB\_CS4.

*kFlexbusMultiplexGroup2\_FB\_TSIZ0* FB\_TSIZ0.

*kFlexbusMultiplexGroup2\_FB\_BE\_31\_24* FB\_BE\_31\_24.

### 17.2.3.9 enum flexbus\_multiplex\_group3\_t

Enumerator

*kFlexbusMultiplexGroup3\_FB\_CS5* FB\_CS5.

*kFlexbusMultiplexGroup3\_FB\_TSIZ1* FB\_TSIZ1.

*kFlexbusMultiplexGroup3\_FB\_BE\_23\_16* FB\_BE\_23\_16.

### 17.2.3.10 enum flexbus\_multiplex\_group4\_t

Enumerator

*kFlexbusMultiplexGroup4\_FB\_TBST* FB\_TBST.

*kFlexbusMultiplexGroup4\_FB\_CS2* FB\_CS2.

*kFlexbusMultiplexGroup4\_FB\_BE\_15\_8* FB\_BE\_15\_8.

## FLEXBUS HAL driver

### 17.2.3.11 enum flexbus\_multiplex\_group5\_t

Enumerator

*kFlexbusMultiplexGroup5\_FB\_TA* FB\_TA.  
*kFlexbusMultiplexGroup5\_FB\_CS3* FB\_CS3.  
*kFlexbusMultiplexGroup5\_FB\_BE\_7\_0* FB\_BE\_7\_0.

## 17.2.4 Function Documentation

### 17.2.4.1 void FLEXBUS\_HAL\_Init ( FB\_Type \* *base* )

Only chip 0 validated and set to known values. Other chips disabled.

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | Flexbus module base number. |
|-------------|-----------------------------|

### 17.2.4.2 void FLEXBUS\_HAL\_Configure ( FB\_Type \* *base*, const flexbus\_user\_config\_t \* *userConfigPtr* )

Parameters

|                      |                                  |
|----------------------|----------------------------------|
| <i>base</i>          | Flexbus module base number.      |
| <i>userConfigPtr</i> | Flexbus input user configuration |

### 17.2.4.3 static void FLEXBUS\_HAL\_WriteAddr ( FB\_Type \* *base*, uint8\_t *chip*, uint16\_t *addr*, uint16\_t *addrMask* ) [inline], [static]

The CSARn registers specify the chip-select base addresses. NOTE: Because the FlexBus module is one of the slaves connected to the crossbar switch, it is only accessible within a certain memory range. Refer to the device memory map for the applicable FlexBus "expansion" address range for which the chip-selects can be active. Set the CSARn registers appropriately.

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | Flexbus module base number. |
|-------------|-----------------------------|

|                 |                                |
|-----------------|--------------------------------|
| <i>chip</i>     | Flexbus chip for validation.   |
| <i>addr</i>     | chip-select base address.      |
| <i>addrMask</i> | chip-select base address mask. |

**17.2.4.4 static void FLEXBUS\_HAL\_SetChipSelectValidCmd ( FB\_Type \* *base*, uint8\_t *chip*, bool *valid* ) [inline], [static]**

Indicates whether the corresponding CSAR, CSMR, and CSCR contents are valid. NOTE: At reset, no chip-select other than FB\_CS0 can be used until the CSMR0[V] is set. Afterward, FB\_CS[5:0] functions as programmed.

Parameters

|              |                                                                                                                                                             |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>  | Flexbus module base number.                                                                                                                                 |
| <i>chip</i>  | Flexbus chip for validation.                                                                                                                                |
| <i>valid</i> | Validation for chip-selects or not. <ul style="list-style-type: none"> <li>• true: chip-select is valid</li> <li>• false: chip-select is invalid</li> </ul> |

**17.2.4.5 static void FLEXBUS\_HAL\_SetWriteProtectionCmd ( FB\_Type \* *base*, uint8\_t *chip*, bool *enable* ) [inline], [static]**

Controls write accesses to the address range in the corresponding CSAR. 0: Read and write accesses are allowed 1: Only read accesses are allowed

Parameters

|               |                                       |
|---------------|---------------------------------------|
| <i>base</i>   | Flexbus module base number.           |
| <i>chip</i>   | Flexbus chip for validation.          |
| <i>enable</i> | Enables or disables write protection. |

**17.2.4.6 static void FLEXBUS\_HAL\_SetBurstWriteCmd ( FB\_Type \* *base*, uint8\_t *chip*, bool *enable* ) [inline], [static]**

Specifies whether burst writes are used for memory associated with each FB\_CS<sub>n</sub>.

0: Break data larger than the specified port size into individual, port-sized, non-burst writes. For example, a longword write to an 8-bit port takes four byte writes. 1: Enables burst write of data larger than the

---

## **FLEXBUS HAL driver**

specified port size, including longword writes to 8 and 16-bit ports, word writes to 8-bit ports, and line writes to 8-, 16-, and 32-bit ports.

## Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | Flexbus module base number.      |
| <i>chip</i>   | Flexbus chip for validation.     |
| <i>enable</i> | Enables or disables burst-write. |

**17.2.4.7 static void FLEXBUS\_HAL\_SetBurstReadCmd ( FB\_Type \* *base*, uint8\_t *chip*, bool *enable* ) [inline], [static]**

Specifies whether burst reads are used for memory associated with each FB\_CS<sub>n</sub>.

0: Data exceeding the specified port size is broken into individual, port-sized, non-burst reads. For example, a longword read from an 8-bit port is broken into four 8-bit reads. 1: Enables data burst reads larger than the specified port size, including longword reads from 8- and 16-bit ports, word reads from 8-bit ports, and line reads from 8, 16-, and 32-bit ports.

## Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | Flexbus module base number.     |
| <i>chip</i>   | Flexbus chip for validation.    |
| <i>enable</i> | Enables or disables burst-read. |

**17.2.4.8 static void FLEXBUS\_HAL\_SetByteModeCmd ( FB\_Type \* *base*, uint8\_t *chip*, bool *enable* ) [inline], [static]**

Specifies the byte enable operation. Certain memories have byte enables that must be asserted during reads and writes. BEM can be set in the relevant CSCR to provide the appropriate mode of byte enable support for these SRAMs.

The FB\_BEn signals are asserted for read and write accesses.

## Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | Flexbus module base number.             |
| <i>chip</i>   | Flexbus chip for validation.            |
| <i>enable</i> | Enables or disables byte-enable support |

## FLEXBUS HAL driver

**17.2.4.9** `static void FLEXBUS_HAL_SetPortSize ( FB_Type * base, uint8_t chip, flexbus_port_size_t size ) [inline], [static]`

Specifies the data port width associated with each chip-select. It determines where data is driven during write cycles and where data is sampled during read cycles.

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | Flexbus module base number.  |
| <i>chip</i> | Flexbus chip for validation. |
| <i>size</i> | Size of port.                |

**17.2.4.10 static void FLEXBUS\_HAL\_SetAutoAcknowledgeCmd ( FB\_Type \* *base*, uint8\_t *chip*, bool *enable* ) [inline], [static]**

Determines the assertion of the internal transfer acknowledge for accesses specified by the chip-select address.

NOTE: If AA is set for a corresponding FB\_CS<sub>n</sub> and the external system asserts an external FB\_TA before the wait-state countdown asserts the internal FB\_TA, the cycle is terminated. Burst cycles increment the address bus between each internal termination. NOTE: This bit must be set if CSPMCR disables FB\_TA.

enable value: 0: No internal FB\_TA is asserted. Cycle is terminated externally 1: Internal transfer acknowledge is asserted as specified by WS

## Parameters

|               |                                       |
|---------------|---------------------------------------|
| <i>base</i>   | Flexbus module base number.           |
| <i>chip</i>   | Flexbus chip for validation.          |
| <i>enable</i> | Enables or disables Auto-acknowledge. |

**17.2.4.11 static void FLEXBUS\_HAL\_SetByteLaneShift ( FB\_Type \* *base*, uint8\_t *chip*, flexbus\_bytelane\_shift\_t *shift* ) [inline], [static]**

Determines if data on FB\_AD appears left-justified or right-justified during the data phase of a FlexBus access.

0: Not shifted. Data is left-justified on FB\_AD. 1: Shifted. Data is right justified on FB\_AD.

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | Flexbus module base number.  |
| <i>chip</i> | Flexbus chip for validation. |

## FLEXBUS HAL driver

|              |                                                |
|--------------|------------------------------------------------|
| <i>shift</i> | Selects left-justified or right-justified data |
|--------------|------------------------------------------------|

### 17.2.4.12 `static void FLEXBUS_HAL_SetWaitStates ( FB_Type * base, uint8_t chip, uint8_t waitStates ) [inline], [static]`

The number of wait states inserted after FB\_CS<sub>n</sub> asserts and before an internal transfer acknowledge is generated (WS = 0 inserts zero wait states, WS = 0x3F inserts 63 wait states).

Parameters

|                   |                              |
|-------------------|------------------------------|
| <i>base</i>       | Flexbus module base number.  |
| <i>chip</i>       | Flexbus chip for validation. |
| <i>waitStates</i> | Defines value of wait states |

### 17.2.4.13 `static void FLEXBUS_HAL_SetWriteAddrHoldOrDeselect ( FB_Type * base, uint8_t chip, flexbus_write_address_hold_t addrHold ) [inline], [static]`

Write address hold or deselect. This field controls the address, data, and attribute hold time after the termination of a write cycle that hits in the chip-select address space. NOTE: The hold time applies only at the end of a transfer. Therefore, during a burst transfer or a transfer to a port size smaller than the transfer size, the hold time is only added after the last bus cycle.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Flexbus module base number.            |
| <i>chip</i>     | Flexbus chip for validation.           |
| <i>addrHold</i> | Value of cycles to hold write address. |

### 17.2.4.14 `static void FLEXBUS_HAL_SetReadAddrHoldOrDeselect ( FB_Type * base, uint8_t chip, flexbus_read_address_hold_t addrHold ) [inline], [static]`

This field controls the address and attribute hold time after the termination during a read cycle that hits in the chip-select address space. NOTE: The hold time applies only at the end of a transfer. Therefore, during a burst transfer or a transfer to a port size smaller than the transfer size, the hold time is only added after the last bus cycle.



## Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>base</i>     | Flexbus module base number.           |
| <i>chip</i>     | Flexbus chip for validation.          |
| <i>addrHold</i> | Value of cycles to hold read address. |

**17.2.4.15 static void FLEXBUS\_HAL\_SetAddrSetup ( FB\_Type \* *base*, uint8\_t *chip*, flexbus\_address\_setup\_t *delay* ) [inline], [static]**

Controls the assertion of the chip-select with respect to assertion of a valid address and attributes. The address and attributes are considered valid at the same time FB\_TS/FB\_ALE asserts.

## Parameters

|              |                              |
|--------------|------------------------------|
| <i>base</i>  | Flexbus module base number.  |
| <i>chip</i>  | Flexbus chip for validation. |
| <i>delay</i> | Value of delay.              |

**17.2.4.16 static void FLEXBUS\_HAL\_SetExtendedAddrLatchCmd ( FB\_Type \* *base*, uint8\_t *chip*, bool *enable* ) [inline], [static]**

Extended address latch enable

0: FB\_TS/FB\_ALE asserts for one bus clock cycle. 1: FB\_TS/FB\_ALE remains asserted until the first positive clock edge after FB\_CS<sub>n</sub> asserts.

## Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | Flexbus module base number.                 |
| <i>chip</i>   | Flexbus chip for validation.                |
| <i>enable</i> | Enables or disables extended address latch. |

**17.2.4.17 static void FLEXBUS\_HAL\_SetSecondaryWaitStateCmd ( FB\_Type \* *base*, uint8\_t *chip*, bool *enable* ) [inline], [static]**

Secondary wait state enable.

0: The WS value inserts wait states before an internal transfer acknowledge is generated for all transfers.  
1: The SWS value inserts wait states before an internal transfer acknowledge is generated for burst transfer secondary terminations.

## FLEXBUS HAL driver

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | Flexbus module base number.    |
| <i>chip</i>   | Flexbus chip for validation.   |
| <i>enable</i> | Enables or disables wait state |

### 17.2.4.18 `static void FLEXBUS_HAL_SetMultiplexControlGroup1 ( FB_Type * base, flexbus_multiplex_group1_t controls ) [inline], [static]`

GROUP1 Controls the multiplexing of the FB\_ALE, FB\_CS1 , and FB\_TS signals.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Flexbus module base number.            |
| <i>controls</i> | Flexbus multiplex settings for Group1. |

Returns

Flexbus status.

### 17.2.4.19 `static flexbus_multiplex_group1_t FLEXBUS_HAL_GetMultiplexControlGroup1 ( FB_Type * base ) [inline], [static]`

GROUP1 Controls the multiplexing of the FB\_ALE, FB\_CS1 , and FB\_TS signals.

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | Flexbus module base number. |
|-------------|-----------------------------|

Returns

Flexbus multiplex settings for Group1.

### 17.2.4.20 `static void FLEXBUS_HAL_SetMultiplexControlGroup2 ( FB_Type * base, flexbus_multiplex_group2_t controls ) [inline], [static]`

GROUP2 Controls the multiplexing of the FB\_TA , FB\_CS3 , and FB\_BE\_7\_0 signals. When GROUP5 is not 0000b, you must write 1b to the CSCR[AA] bit. Otherwise, the bus hangs during a transfer.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Flexbus module base number.            |
| <i>controls</i> | Flexbus multiplex settings for Group2. |

Returns

Flexbus status.

**17.2.4.21 static flexbus\_multiplex\_group2\_t FLEXBUS\_HAL\_GetMultiplexControlGroup2 ( FB\_Type \* *base* ) [inline], [static]**

GROUP2 Controls the multiplexing of the FB\_TA , FB\_CS3 , and FB\_BE\_7\_0 signals. When GROUP5 is not 0000b, you must write 1b to the CSCR[AA] bit. Otherwise, the bus hangs during a transfer.

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | Flexbus module base number. |
|-------------|-----------------------------|

Returns

Flexbus multiplex settings for Group2.

**17.2.4.22 static void FLEXBUS\_HAL\_SetMultiplexControlGroup3 ( FB\_Type \* *base*, flexbus\_multiplex\_group3\_t *controls* ) [inline], [static]**

GROUP3 Controls the multiplexing of the FB\_CS4 , FB\_TSIZ0, and FB\_BE\_31\_24 signals.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Flexbus module base number.            |
| <i>controls</i> | Flexbus multiplex settings for Group3. |

Returns

Flexbus status.

**17.2.4.23 static flexbus\_multiplex\_group3\_t FLEXBUS\_HAL\_GetMultiplexControlGroup3 ( FB\_Type \* *base* ) [inline], [static]**

GROUP3 Controls the multiplexing of the FB\_CS4 , FB\_TSIZ0, and FB\_BE\_31\_24 signals.

## FLEXBUS HAL driver

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | Flexbus module base number. |
|-------------|-----------------------------|

Returns

Flexbus multiplex settings for Group3.

**17.2.4.24 static void FLEXBUS\_HAL\_SetMultiplexControlGroup4 ( FB\_Type \* *base*, flexbus\_multiplex\_group4\_t *controls* ) [inline], [static]**

GROUP4 Controls the multiplexing of the FB\_TBST, FB\_CS2, and FB\_BE\_15\_8 signals.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Flexbus module base number.            |
| <i>controls</i> | Flexbus multiplex settings for Group4. |

Returns

Flexbus status.

**17.2.4.25 static flexbus\_multiplex\_group4\_t FLEXBUS\_HAL\_GetMultiplexControlGroup4 ( FB\_Type \* *base* ) [inline], [static]**

GROUP4 Controls the multiplexing of the FB\_TBST, FB\_CS2, and FB\_BE\_15\_8 signals.

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | Flexbus module base number. |
|-------------|-----------------------------|

Returns

Flexbus multiplex settings for Group4.

**17.2.4.26 static void FLEXBUS\_HAL\_SetMultiplexControlGroup5 ( FB\_Type \* *base*, flexbus\_multiplex\_group5\_t *controls* ) [inline], [static]**

GROUP5 Controls the multiplexing of the FB\_TA, FB\_CS3, and FB\_BE\_7\_0 signals.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Flexbus module base number.            |
| <i>controls</i> | Flexbus multiplex settings for Group5. |

Returns

Flexbus status.

**17.2.4.27 static flexbus\_multiplex\_group5\_t FLEXBUS\_HAL\_GetMultiplexControlGroup5 ( FB\_Type \* *base* ) [inline], [static]**

GROUP5 Controls the multiplexing of the FB\_TA, FB\_CS3, and FB\_BE\_7\_0 signals.

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | Flexbus module base number. |
|-------------|-----------------------------|

Returns

Flexbus multiplex settings for Group5.

## FLEXBUS Peripheral Driver

### 17.3 FLEXBUS Peripheral Driver

#### 17.3.1 Overview

The section describes the programming interface of the FLEXBUS Peripheral driver. The FLEXBUS driver provides an easy way to initialize and configure the FLEXBUS.

#### 17.3.2 FLEXBUS Overview

The FLEXBUS (external bus interface) is a hardware module that provides memory expansion and a connection to external peripheral with a parallel bus. It can be directly connected to the slave-only devices such as:

- External ROMs
- Flash memories
- Programmable logic devices
- Other simple target (slave) devices

#### 17.3.3 FLEXBUS Initialization

To initialize the FLEXBUS module, call the [FLEXBUS\\_DRV\\_Init\(\)](#) function and pass in the instance number and the user configuration structure. This function automatically enables the FLEXBUS module and clock.

This example code shows how to initialize and configure the FLEXBUS for MRAM purpose:

```

/* Define configuration.*/
flexbus_user_config_t fb_config;
/* Clear user configuration struct first.*/
memset(&fb_config, 0x0, sizeof(flexbus_user_config_t));
fb_config.chip = 0;
fb_config.baseAddress = 0x60000000; /* Configure base address.*/
fb_config.portSize = kFlexbus1byte; /* Port size of transfer.*/
fb_config.autoAcknowledge = true; /* Enable auto acknowledge.*/
fb_config.waitStates = 0x2; /* Set wait state.*/
fb_config.baseAddressMask = 0x7; /* Configure base address mask.*/

/* Initialize FLEXBUS.*/
FLEXBUS_DRV_Init(instance, &fb_config);

```

#### 17.3.4 FLEXBUS De-initialize

To shut down the FLEXBUS module, call the [FLEXBUS\\_DRV\\_Deinit\(\)](#) function and pass in the instance number.

```

FLEXBUS_DRV_Deinit(instance);

```

## Functions

- `flexbus_status_t FLEXBUS_DRV_Init (uint32_t instance, const flexbus_user_config_t *fb_config)`  
*Initializes the FlexBus driver.*
- `flexbus_status_t FLEXBUS_DRV_Deinit (uint32_t instance)`  
*Shuts down the FlexBus driver.*

## Variables

- `FB_Type *const g_fbBase []`  
*Table of base addresses for FlexBus instances.*

### 17.3.5 Function Documentation

#### 17.3.5.1 flexbus\_status\_t FLEXBUS\_DRV\_Init ( uint32\_t instance, const flexbus\_user\_config\_t \* fb\_config )

Parameters

|                  |                                         |
|------------------|-----------------------------------------|
| <i>instance</i>  | The FlexBus peripheral instance number. |
| <i>fb_config</i> | FlexBus input user configuration        |

#### 17.3.5.2 flexbus\_status\_t FLEXBUS\_DRV\_Deinit ( uint32\_t instance )

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>instance</i> | The FlexBus peripheral instance number. |
|-----------------|-----------------------------------------|

### 17.3.6 Variable Documentation

#### 17.3.6.1 FB\_Type\* const g\_fbBase[]





---

## Chapter 18

# Controller Area Network (FlexCAN)

### 18.1 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](#)

## FlexCAN HAL driver

### 18.2 FlexCAN HAL driver

#### 18.2.1 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\\_buserr\\_counter\\_t](#)  
*FlexCAN bus error counters. [More...](#)*
- struct [flexcan\\_msgbuff\\_code\\_status\\_t](#)  
*FlexCAN Message Buffer code and status for transmit and receive. [More...](#)*
- struct [flexcan\\_msgbuff\\_t](#)  
*FlexCAN message buffer structure. [More...](#)*
- struct [flexcan\\_time\\_segment\\_t](#)  
*FlexCAN timing related structures. [More...](#)*

#### Enumerations

- enum [\\_flexcan\\_constants](#) { [kFlexCanMessageSize](#) = 8 }
  - enum [\\_flexcan\\_err\\_status](#) {  
[kFlexCanRxWrn](#) = 0x0080U,  
[kFlexCanTxWrn](#) = 0x0100U,  
[kFlexCanStfErr](#) = 0x0200U,  
[kFlexCanFrmErr](#) = 0x0400U,  
[kFlexCanCrcErr](#) = 0x0800U,  
[kFlexCanAckErr](#) = 0x1000U,  
[kFlexCanBit0Err](#) = 0x2000U,  
[kFlexCanBit1Err](#) = 0x4000U }
  - enum [flexcan\\_status\\_t](#)  
*FlexCAN status return codes.*
  - enum [flexcan\\_operation\\_modes\\_t](#) {  
[kFlexCanNormalMode](#),  
[kFlexCanListenOnlyMode](#),  
[kFlexCanLoopBackMode](#),  
[kFlexCanFreezeMode](#),  
[kFlexCanDisableMode](#) }
  - enum [flexcan\\_msgbuff\\_code\\_rx\\_t](#) {
- The Status enum is used to report current status of the FlexCAN interface.*
- FlexCAN operation modes.*

```

kFlexCanRXInactive = 0x0,
kFlexCanRXFull = 0x2,
kFlexCanRXEmpty = 0x4,
kFlexCanRXOverrun = 0x6,
kFlexCanRXBusy = 0x8,
kFlexCanRXRanswer = 0xA,
kFlexCanRXNotUsed = 0xF }

```

*FlexCAN message buffer CODE for Rx buffers.*

- enum flexcan\_msgbuff\_code\_tx\_t {

```

kFlexCanTXInactive = 0x08,
kFlexCanTXAbort = 0x09,
kFlexCanTXData = 0x0C,
kFlexCanTXRemote = 0x1C,
kFlexCanTXAnswer = 0x0E,
kFlexCanTXNotUsed = 0xF }

```

*FlexCAN message buffer CODE FOR Tx buffers.*

- enum flexcan\_msgbuff\_transmission\_type\_t {

```

kFlexCanMBStatusTypeTX,
kFlexCanMBStatusTypeTXRemote,
kFlexCanMBStatusTypeRX,
kFlexCanMBStatusTypeRXRemote,
kFlexCanMBStatusTypeRXTXRemote }

```

*FlexCAN message buffer transmission types.*

- enum flexcan\_rx\_fifo\_id\_element\_format\_t {

```

kFlexCanRxFifoIdElementFormatA,
kFlexCanRxFifoIdElementFormatB,
kFlexCanRxFifoIdElementFormatC,
kFlexCanRxFifoIdElementFormatD }

```
- 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` {  
`kFlexCanRxMaskGlobal`,  
`kFlexCanRxMaskIndividual` }  
*FlexCAN RX mask type.*
- enum `flexcan_msgbuff_id_type_t` {  
`kFlexCanMsgIdStd`,  
`kFlexCanMsgIdExt` }  
*FlexCAN Message Buffer ID type.*
- enum `flexcan_clk_source_t` {  
`kFlexCanClkSourceOsc`,  
`kFlexCanClkSourceIpbuss` }  
*FlexCAN clock source.*
- enum `flexcan_int_type_t` {  
`kFlexCanIntRxwarning` = `CAN_CTRL1_RWRNMSK_MASK`,  
`kFlexCanIntTxwarning` = `CAN_CTRL1_TWRNMSK_MASK`,  
`kFlexCanIntErr` = `CAN_CTRL1_ERRMSK_MASK`,  
`kFlexCanIntBusoff` = `CAN_CTRL1_BOFFMSK_MASK`,  
`kFlexCanIntWakeup` = `CAN_MCR_WAKMSK_MASK` }  
*FlexCAN error interrupt types.*

## Configuration

- `flexcan_status_t FLEXCAN_HAL_Enable` (`CAN_Type *base`)  
*Enables FlexCAN controller.*
- `flexcan_status_t FLEXCAN_HAL_Disable` (`CAN_Type *base`)  
*Disables FlexCAN controller.*
- `flexcan_status_t FLEXCAN_HAL_SelectClock` (`CAN_Type *base`, `flexcan_clk_source_t clk`)  
*Selects the clock source for FlexCAN.*
- static bool `FLEXCAN_HAL_GetClock` (`CAN_Type *base`)  
*Reads the clock source for FlexCAN Protocol Engine (PE).*
- `flexcan_status_t FLEXCAN_HAL_Init` (`CAN_Type *base`)  
*Initializes the FlexCAN controller.*
- void `FLEXCAN_HAL_SetTimeSegments` (`CAN_Type *base`, `flexcan_time_segment_t *timeSeg`)  
*Sets the FlexCAN time segments for setting up bit rate.*
- void `FLEXCAN_HAL_GetTimeSegments` (`CAN_Type *base`, `flexcan_time_segment_t *timeSeg`)  
*Gets the FlexCAN time segments to calculate the bit rate.*
- void `FLEXCAN_HAL_ExitFreezeMode` (`CAN_Type *base`)  
*Unfreezes the FlexCAN module.*
- void `FLEXCAN_HAL_EnterFreezeMode` (`CAN_Type *base`)  
*Freezes the FlexCAN module.*
- `flexcan_status_t FLEXCAN_HAL_SetOperationMode` (`CAN_Type *base`, `flexcan_operation_modes_t mode`)  
*Set operation mode.*
- `flexcan_status_t FLEXCAN_HAL_ExitOperationMode` (`CAN_Type *base`, `flexcan_operation_modes_t mode`)  
*Exit operation mode.*

## Data transfer

- [flexcan\\_status\\_t FLEXCAN\\_HAL\\_SetTxMsgBuff](#) (CAN\_Type \*base, uint32\_t msgBuffIdx, flexcan\_msgbuff\_code\_status\_t \*cs, uint32\_t msgId, uint8\_t \*msgData)  
*Sets the FlexCAN message buffer fields for transmitting.*
- [flexcan\\_status\\_t FLEXCAN\\_HAL\\_SetRxMsgBuff](#) (CAN\_Type \*base, uint32\_t msgBuffIdx, flexcan\_msgbuff\_code\_status\_t \*cs, uint32\_t msgId)  
*Sets the FlexCAN message buffer fields for receiving.*
- [flexcan\\_status\\_t FLEXCAN\\_HAL\\_GetMsgBuff](#) (CAN\_Type \*base, uint32\_t msgBuffIdx, flexcan\_msgbuff\_t \*msgBuff)  
*Gets the FlexCAN message buffer fields.*
- [flexcan\\_status\\_t FLEXCAN\\_HAL\\_LockRxMsgBuff](#) (CAN\_Type \*base, uint32\_t msgBuffIdx)  
*Locks the FlexCAN Rx message buffer.*
- static uint32\_t [FLEXCAN\\_HAL\\_UnlockRxMsgBuff](#) (CAN\_Type \*base)  
*Unlocks the FlexCAN Rx message buffer.*
- void [FLEXCAN\\_HAL\\_EnableRxFifo](#) (CAN\_Type \*base, uint32\_t numOfFilters)  
*Enables the Rx FIFO.*
- void [FLEXCAN\\_HAL\\_DisableRxFifo](#) (CAN\_Type \*base)  
*Disables the Rx FIFO.*
- void [FLEXCAN\\_HAL\\_SetRxFifoFilterNum](#) (CAN\_Type \*base, uint32\_t number)  
*Sets the number of the Rx FIFO filters.*
- void [FLEXCAN\\_HAL\\_SetMaxMsgBuffNum](#) (CAN\_Type \*base, uint32\_t maxMsgBuffNum)  
*Sets the maximum number of Message Buffers.*
- [flexcan\\_status\\_t FLEXCAN\\_HAL\\_SetRxFifoFilter](#) (CAN\_Type \*base, flexcan\_rx\_fifo\_id\_element\_format\_t idFormat, flexcan\_id\_table\_t \*idFilterTable)  
*Sets the FlexCAN Rx FIFO fields.*
- [flexcan\\_status\\_t FLEXCAN\\_HAL\\_ReadRxFifo](#) (CAN\_Type \*base, flexcan\_msgbuff\_t \*rxFifo)  
*Gets the FlexCAN Rx FIFO data.*

## Interrupts

- [flexcan\\_status\\_t FLEXCAN\\_HAL\\_SetMsgBuffIntCmd](#) (CAN\_Type \*base, uint32\_t msgBuffIdx, bool enable)  
*Enables/Disables the FlexCAN Message Buffer interrupt.*
- void [FLEXCAN\\_HAL\\_SetErrIntCmd](#) (CAN\_Type \*base, flexcan\_int\_type\_t errType, bool enable)  
*Enables error interrupt of the FlexCAN module.*

## Status

- static uint32\_t [FLEXCAN\\_HAL\\_GetFreezeAck](#) (CAN\_Type \*base)  
*Gets the value of FlexCAN freeze ACK.*
- uint8\_t [FLEXCAN\\_HAL\\_GetMsgBuffIntStatusFlag](#) (CAN\_Type \*base, uint32\_t msgBuffIdx)  
*Gets the individual FlexCAN MB interrupt flag.*
- static uint32\_t [FLEXCAN\\_HAL\\_GetAllMsgBuffIntStatusFlag](#) (CAN\_Type \*base)  
*Gets all FlexCAN Message Buffer interrupt flags.*
- static void [FLEXCAN\\_HAL\\_ClearMsgBuffIntStatusFlag](#) (CAN\_Type \*base, uint32\_t flag)  
*Clears the interrupt flag of the message buffers.*
- void [FLEXCAN\\_HAL\\_GetErrCounter](#) (CAN\_Type \*base, flexcan\_buserr\_counter\_t \*errCount)

## FlexCAN HAL driver

- Gets the transmit error counter and receives the error counter.*

  - static uint32\_t **FLEXCAN\_HAL\_GetErrStatus** (CAN\_Type \*base)

*Gets error and status.*
- void **FLEXCAN\_HAL\_ClearErrIntStatusFlag** (CAN\_Type \*base)

*Clears all other interrupts in ERRSTAT register (Error, Busoff, Wakeup).*

## Mask

- void **FLEXCAN\_HAL\_SetRxMaskType** (CAN\_Type \*base, flexcan\_rx\_mask\_type\_t type)

*Sets the Rx masking type.*
- void **FLEXCAN\_HAL\_SetRxFifoGlobalStdMask** (CAN\_Type \*base, uint32\_t stdMask)

*Sets the FlexCAN RX FIFO global standard mask.*
- void **FLEXCAN\_HAL\_SetRxFifoGlobalExtMask** (CAN\_Type \*base, uint32\_t extMask)

*Sets the FlexCAN Rx FIFO global extended mask.*
- flexcan\_status\_t **FLEXCAN\_HAL\_SetRxIndividualStdMask** (CAN\_Type \*base, uint32\_t msg-BuffIdx, uint32\_t stdMask)

*Sets the FlexCAN Rx individual standard mask for ID filtering in the Rx MBs and the Rx FIFO.*
- flexcan\_status\_t **FLEXCAN\_HAL\_SetRxIndividualExtMask** (CAN\_Type \*base, uint32\_t msg-BuffIdx, uint32\_t extMask)

*Sets the FlexCAN Rx individual extended mask for ID filtering in the Rx Message Buffers and the Rx FIFO.*
- void **FLEXCAN\_HAL\_SetRxMsgBuffGlobalStdMask** (CAN\_Type \*base, uint32\_t stdMask)

*Sets the FlexCAN Rx Message Buffer global standard mask.*
- void **FLEXCAN\_HAL\_SetRxMsgBuff14StdMask** (CAN\_Type \*base, uint32\_t stdMask)

*Sets the FlexCAN RX Message Buffer BUF14 standard mask.*
- void **FLEXCAN\_HAL\_SetRxMsgBuff15StdMask** (CAN\_Type \*base, uint32\_t stdMask)

*Sets the FlexCAN Rx Message Buffer BUF15 standard mask.*
- void **FLEXCAN\_HAL\_SetRxMsgBuffGlobalExtMask** (CAN\_Type \*base, uint32\_t extMask)

*Sets the FlexCAN RX Message Buffer global extended mask.*
- void **FLEXCAN\_HAL\_SetRxMsgBuff14ExtMask** (CAN\_Type \*base, uint32\_t extMask)

*Sets the FlexCAN RX Message Buffer BUF14 extended mask.*
- void **FLEXCAN\_HAL\_SetRxMsgBuff15ExtMask** (CAN\_Type \*base, uint32\_t extMask)

*Sets the FlexCAN RX MB BUF15 extended mask.*
- static uint32\_t **FLEXCAN\_HAL\_GetRxFifoHitIdAcceptanceFilter** (CAN\_Type \*base)

*Gets the FlexCAN ID acceptance filter hit indicator on Rx FIFO.*

## 18.2.2 Data Structure Documentation

### 18.2.2.1 struct flexcan\_id\_table\_t

#### Data Fields

- bool **isRemoteFrame**

*Remote frame.*
- bool **isExtendedFrame**

*Extended frame.*
- uint32\_t \* **idFilter**

*Rx FIFO ID filter elements.*

### 18.2.2.2 struct flexcan\_buserr\_counter\_t

#### Data Fields

- uint16\_t [txerr](#)  
*Transmit error counter.*
- uint16\_t [rxerr](#)  
*Receive error counter.*

### 18.2.2.3 struct flexcan\_msgbuff\_code\_status\_t

#### Data Fields

- uint32\_t [code](#)  
*MB code for TX or RX buffers.*
- [flexcan\\_msgbuff\\_id\\_type\\_t](#) [msgIdType](#)  
*Type of message ID (standard or extended)*
- uint32\_t [dataLen](#)  
*Length of Data in Bytes.*

#### 18.2.2.3.0.29 Field Documentation

##### 18.2.2.3.0.29.1 uint32\_t flexcan\_msgbuff\_code\_status\_t::code

Defined by [flexcan\\_mb\\_code\\_rx\\_t](#) and [flexcan\\_mb\\_code\\_tx\\_t](#)

### 18.2.2.4 struct flexcan\_msgbuff\_t

#### Data Fields

- uint32\_t [cs](#)  
*Code and Status.*
- uint32\_t [msgId](#)  
*Message Buffer ID.*
- uint8\_t [data](#) [[kFlexCanMessageSize](#)]  
*Bytes of the FlexCAN message.*

### 18.2.2.5 struct flexcan\_time\_segment\_t

#### Data Fields

- uint32\_t [propSeg](#)  
*Propagation segment.*
- uint32\_t [phaseSeg1](#)  
*Phase segment 1.*
- uint32\_t [phaseSeg2](#)  
*Phase segment 2.*
- uint32\_t [preDivider](#)

## FlexCAN HAL driver

- *Clock pre divider.*  
 • `uint32_t rJumpwidth`  
*Resync jump width.*

### 18.2.3 Enumeration Type Documentation

#### 18.2.3.1 enum `_flexcan_constants`

Enumerator

*kFlexCanMessageSize* FlexCAN message buffer data size in bytes.

#### 18.2.3.2 enum `_flexcan_err_status`

Enumerator

*kFlexCanRxWrn* Reached warning level for RX errors.

*kFlexCanTxWrn* Reached warning level for TX errors.

*kFlexCanStfErr* Stuffing Error.

*kFlexCanFrmErr* Form Error.

*kFlexCanCrcErr* Cyclic Redundancy Check Error.

*kFlexCanAckErr* Received no ACK on transmission.

*kFlexCanBit0Err* Unable to send dominant bit.

*kFlexCanBit1Err* Unable to send recessive bit.

#### 18.2.3.3 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.

#### 18.2.3.4 enum `flexcan_msgbuff_code_rx_t`

Enumerator

*kFlexCanRXInactive* MB is not active.

*kFlexCanRXFull* MB is full.

*kFlexCanRXEmpty* MB is active and empty.

*kFlexCanRXOverrun* MB is overwritten into a full buffer.



***kFlexCanRXBusy*** FlexCAN is updating the contents of the MB.

***kFlexCanRXRanswer*** The CPU must not access the MB. A frame was configured to recognize a Remote Request Frame

***kFlexCanRXNotUsed*** and transmit a Response Frame in return. Not used

### 18.2.3.5 enum flexcan\_msgbuff\_code\_tx\_t

Enumerator

***kFlexCanTXInactive*** MB is not active.

***kFlexCanTXAbort*** MB is aborted.

***kFlexCanTXData*** MB is a TX Data Frame(MB RTR must be 0).

***kFlexCanTXRemote*** MB is a TX Remote Request Frame (MB RTR must be 1).

***kFlexCanTXTanswer*** MB is a TX Response Request Frame from.

***kFlexCanTXNotUsed*** an incoming Remote Request Frame. Not used

### 18.2.3.6 enum flexcan\_msgbuff\_transmission\_type\_t

Enumerator

***kFlexCanMBStatusTypeTX*** Transmit MB.

***kFlexCanMBStatusTypeTXRemote*** Transmit remote request MB.

***kFlexCanMBStatusTypeRX*** Receive MB.

***kFlexCanMBStatusTypeRXRemote*** Receive remote request MB.

***kFlexCanMBStatusTypeRXTXRemote*** FlexCAN remote frame receives remote request and. transmits MB.

### 18.2.3.7 enum flexcan\_rx\_fifo\_id\_element\_format\_t

Enumerator

***kFlexCanRxFifoIdElementFormatA*** One full ID (standard and extended) per ID Filter Table.

***kFlexCanRxFifoIdElementFormatB*** element. Two full standard IDs or two partial 14-bit (standard and

***kFlexCanRxFifoIdElementFormatC*** extended) IDs per ID Filter Table element. Four partial 8-bit Standard IDs per ID Filter Table

***kFlexCanRxFifoIdElementFormatD*** element. All frames rejected.

### 18.2.3.8 enum flexcan\_rx\_fifo\_id\_filter\_num\_t

Enumerator

***kFlexCanRxFifoIDFilters\_8*** 8 Rx FIFO Filters.

## FlexCAN HAL driver

|                                    |                      |
|------------------------------------|----------------------|
| <i>kFlexCanRxFifoIDFilters_16</i>  | 16 Rx FIFO Filters.  |
| <i>kFlexCanRxFifoIDFilters_24</i>  | 24 Rx FIFO Filters.  |
| <i>kFlexCanRxFifoIDFilters_32</i>  | 32 Rx FIFO Filters.  |
| <i>kFlexCanRxFifoIDFilters_40</i>  | 40 Rx FIFO Filters.  |
| <i>kFlexCanRxFifoIDFilters_48</i>  | 48 Rx FIFO Filters.  |
| <i>kFlexCanRxFifoIDFilters_56</i>  | 56 Rx FIFO Filters.  |
| <i>kFlexCanRxFifoIDFilters_64</i>  | 64 Rx FIFO Filters.  |
| <i>kFlexCanRxFifoIDFilters_72</i>  | 72 Rx FIFO Filters.  |
| <i>kFlexCanRxFifoIDFilters_80</i>  | 80 Rx FIFO Filters.  |
| <i>kFlexCanRxFifoIDFilters_88</i>  | 88 Rx FIFO Filters.  |
| <i>kFlexCanRxFifoIDFilters_96</i>  | 96 Rx FIFO Filters.  |
| <i>kFlexCanRxFifoIDFilters_104</i> | 104 Rx FIFO Filters. |
| <i>kFlexCanRxFifoIDFilters_112</i> | 112 Rx FIFO Filters. |
| <i>kFlexCanRxFifoIDFilters_120</i> | 120 Rx FIFO Filters. |
| <i>kFlexCanRxFifoIDFilters_128</i> | 128 Rx FIFO Filters. |

### 18.2.3.9 enum flexcan\_rx\_mask\_type\_t

Enumerator

|                                 |                     |
|---------------------------------|---------------------|
| <i>kFlexCanRxMaskGlobal</i>     | Rx global mask.     |
| <i>kFlexCanRxMaskIndividual</i> | Rx individual mask. |

### 18.2.3.10 enum flexcan\_msgbuff\_id\_type\_t

Enumerator

|                         |              |
|-------------------------|--------------|
| <i>kFlexCanMsgIdStd</i> | Standard ID. |
| <i>kFlexCanMsgIdExt</i> | Extended ID. |

### 18.2.3.11 enum flexcan\_clk\_source\_t

Enumerator

|                              |                   |
|------------------------------|-------------------|
| <i>kFlexCanClkSourceOsc</i>  | Oscillator clock. |
| <i>kFlexCanClkSourceIpbu</i> | Peripheral clock. |

### 18.2.3.12 enum flexcan\_int\_type\_t

Enumerator

|                             |                       |
|-----------------------------|-----------------------|
| <i>kFlexCanIntRxwarning</i> | RX warning interrupt. |
|-----------------------------|-----------------------|

*kFlexCanIntTxwarning* TX warning interrupt.  
*kFlexCanIntErr* Error interrupt.  
*kFlexCanIntBusoff* Bus off interrupt.  
*kFlexCanIntWakeup* Wake up interrupt.

## 18.2.4 Function Documentation

### 18.2.4.1 flexcan\_status\_t FLEXCAN\_HAL\_Enable ( CAN\_Type \* base )

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
|-------------|--------------------------|

Returns

0 if successful; non-zero failed

### 18.2.4.2 flexcan\_status\_t FLEXCAN\_HAL\_Disable ( CAN\_Type \* base )

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
|-------------|--------------------------|

Returns

0 if successful; non-zero failed

### 18.2.4.3 flexcan\_status\_t FLEXCAN\_HAL\_SelectClock ( CAN\_Type \* base, flexcan\_clk\_source\_t clk )

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
| <i>clk</i>  | The FlexCAN clock source |

Returns

0 if successful; non-zero failed

### 18.2.4.4 static bool FLEXCAN\_HAL\_GetClock ( CAN\_Type \* base ) [inline], [static]

## FlexCAN HAL driver

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
|-------------|--------------------------|

Returns

0: if clock source is oscillator clock, 1: if clock source is peripheral clock

### 18.2.4.5 flexcan\_status\_t FLEXCAN\_HAL\_Init ( CAN\_Type \* *base* )

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
|-------------|--------------------------|

Returns

0 if successful; non-zero failed

### 18.2.4.6 void FLEXCAN\_HAL\_SetTimeSegments ( CAN\_Type \* *base*, flexcan\_time\_segment\_t \* *timeSeg* )

Parameters

|                |                                                               |
|----------------|---------------------------------------------------------------|
| <i>base</i>    | The FlexCAN base address                                      |
| <i>timeSeg</i> | FlexCAN time segments, which need to be set for the bit rate. |

Returns

0 if successful; non-zero failed

### 18.2.4.7 void FLEXCAN\_HAL\_GetTimeSegments ( CAN\_Type \* *base*, flexcan\_time\_segment\_t \* *timeSeg* )

Parameters

---

|                |                                         |
|----------------|-----------------------------------------|
| <i>base</i>    | The FlexCAN base address                |
| <i>timeSeg</i> | FlexCAN time segments read for bit rate |

Returns

0 if successful; non-zero failed

#### 18.2.4.8 void FLEXCAN\_HAL\_ExitFreezeMode ( CAN\_Type \* *base* )

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
|-------------|--------------------------|

Returns

0 if successful; non-zero failed.

#### 18.2.4.9 void FLEXCAN\_HAL\_EnterFreezeMode ( CAN\_Type \* *base* )

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
|-------------|--------------------------|

#### 18.2.4.10 flexcan\_status\_t FLEXCAN\_HAL\_SetOperationMode ( CAN\_Type \* *base*, flexcan\_operation\_modes\_t *mode* )

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
| <i>mode</i> | Set an operation mode    |

Returns

0 if successful; non-zero failed.

#### 18.2.4.11 flexcan\_status\_t FLEXCAN\_HAL\_ExitOperationMode ( CAN\_Type \* *base*, flexcan\_operation\_modes\_t *mode* )

## FlexCAN HAL driver

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
| <i>mode</i> | Exit An operation mode   |

### Returns

0 if successful; non-zero failed.

#### 18.2.4.12 flexcan\_status\_t FLEXCAN\_HAL\_SetTxMsgBuff ( CAN\_Type \* *base*, uint32\_t *msgBuffIdx*, flexcan\_msgbuff\_code\_status\_t \* *cs*, uint32\_t *msgId*, uint8\_t \* *msgData* )

### Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | The FlexCAN base address      |
| <i>msgBuffIdx</i> | Index of the message buffer   |
| <i>cs</i>         | CODE/status values (TX)       |
| <i>msgId</i>      | ID of the message to transmit |
| <i>msgData</i>    | Bytes of the FlexCAN message  |

### Returns

0 if successful; non-zero failed

#### 18.2.4.13 flexcan\_status\_t FLEXCAN\_HAL\_SetRxMsgBuff ( CAN\_Type \* *base*, uint32\_t *msgBuffIdx*, flexcan\_msgbuff\_code\_status\_t \* *cs*, uint32\_t *msgId* )

### Parameters

|                   |                              |
|-------------------|------------------------------|
| <i>base</i>       | The FlexCAN base address     |
| <i>msgBuffIdx</i> | Index of the message buffer  |
| <i>cs</i>         | CODE/status values (RX)      |
| <i>msgId</i>      | ID of the message to receive |

### Returns

0 if successful; non-zero failed

**18.2.4.14** `flexcan_status_t FLEXCAN_HAL_GetMsgBuff ( CAN_Type * base, uint32_t msgBuffIdx, flexcan_msgbuff_t * msgBuff )`

## FlexCAN HAL driver

### Parameters

|                   |                                  |
|-------------------|----------------------------------|
| <i>base</i>       | The FlexCAN base address         |
| <i>msgBuffIdx</i> | Index of the message buffer      |
| <i>msgBuff</i>    | The fields of the message buffer |

### Returns

0 if successful; non-zero failed

#### 18.2.4.15 flexcan\_status\_t FLEXCAN\_HAL\_LockRxMsgBuff ( CAN\_Type \* *base*, uint32\_t *msgBuffIdx* )

### Parameters

|                   |                             |
|-------------------|-----------------------------|
| <i>base</i>       | The FlexCAN base address    |
| <i>msgBuffIdx</i> | Index of the message buffer |

### Returns

0 if successful; non-zero failed

#### 18.2.4.16 static uint32\_t FLEXCAN\_HAL\_UnlockRxMsgBuff ( CAN\_Type \* *base* ) [inline], [static]

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
|-------------|--------------------------|

### Returns

0 if successful; non-zero failed

#### 18.2.4.17 void FLEXCAN\_HAL\_EnableRxFifo ( CAN\_Type \* *base*, uint32\_t *numOfFilters* )



Parameters

|                     |                               |
|---------------------|-------------------------------|
| <i>base</i>         | The FlexCAN base address      |
| <i>numOfFilters</i> | The number of Rx FIFO filters |

**18.2.4.18 void FLEXCAN\_HAL\_DisableRxFifo ( CAN\_Type \* *base* )**

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
|-------------|--------------------------|

**18.2.4.19 void FLEXCAN\_HAL\_SetRxFifoFilterNum ( CAN\_Type \* *base*, uint32\_t *number* )**

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | The FlexCAN base address      |
| <i>number</i> | The number of Rx FIFO filters |

**18.2.4.20 void FLEXCAN\_HAL\_SetMaxMsgBuffNum ( CAN\_Type \* *base*, uint32\_t *maxMsgBuffNum* )**

Parameters

|                      |                                   |
|----------------------|-----------------------------------|
| <i>base</i>          | The FlexCAN base address          |
| <i>maxMsgBuffNum</i> | Maximum number of message buffers |

**18.2.4.21 flexcan\_status\_t FLEXCAN\_HAL\_SetRxFifoFilter ( CAN\_Type \* *base*, flexcan\_rx\_fifo\_id\_element\_format\_t *idFormat*, flexcan\_id\_table\_t \* *idFilterTable* )**

Parameters

|                      |                                                                                 |
|----------------------|---------------------------------------------------------------------------------|
| <i>base</i>          | The FlexCAN base address                                                        |
| <i>idFormat</i>      | The format of the Rx FIFO ID Filter Table Elements                              |
| <i>idFilterTable</i> | The ID filter table elements which contain RTR bit, IDE bit, and RX message ID. |

## FlexCAN HAL driver

Returns

0 if successful; non-zero failed.

**18.2.4.22** `flexcan_status_t FLEXCAN_HAL_ReadRxFifo ( CAN_Type * base, flexcan_msgbuff_t * rxFifo )`

Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | The FlexCAN base address      |
| <i>rxFifo</i> | The FlexCAN receive FIFO data |

Returns

0 if successful; non-zero failed.

**18.2.4.23** `flexcan_status_t FLEXCAN_HAL_SetMsgBuffIntCmd ( CAN_Type * base, uint32_t msgBuffIdx, bool enable )`

Parameters

|                   |                             |
|-------------------|-----------------------------|
| <i>base</i>       | The FlexCAN base address    |
| <i>msgBuffIdx</i> | Index of the message buffer |
| <i>enable</i>     | choose enable or disable    |

Returns

0 if successful; non-zero failed

**18.2.4.24** `void FLEXCAN_HAL_SetErrIntCmd ( CAN_Type * base, flexcan_int_type_t errType, bool enable )`

Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | The FlexCAN base address |
| <i>errType</i> | The interrupt type       |
| <i>enable</i>  | choose enable or disable |

**18.2.4.25** `static uint32_t FLEXCAN_HAL_GetFreezeAck ( CAN_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
|-------------|--------------------------|

Returns

freeze ACK state (1-freeze mode, 0-not in freeze mode).

**18.2.4.26** `uint8_t FLEXCAN_HAL_GetMsgBuffIntStatusFlag ( CAN_Type * base, uint32_t msgBuffIdx )`

Parameters

|                   |                             |
|-------------------|-----------------------------|
| <i>base</i>       | The FlexCAN base address    |
| <i>msgBuffIdx</i> | Index of the message buffer |

Returns

the individual Message Buffer interrupt flag (0 and 1 are the flag value)

**18.2.4.27** `static uint32_t FLEXCAN_HAL_GetAllMsgBuffIntStatusFlag ( CAN_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
|-------------|--------------------------|

Returns

all MB interrupt flags

**18.2.4.28** `static void FLEXCAN_HAL_ClearMsgBuffIntStatusFlag ( CAN_Type * base, uint32_t flag ) [inline], [static]`

## FlexCAN HAL driver

### Parameters

|             |                                                          |
|-------------|----------------------------------------------------------|
| <i>base</i> | The FlexCAN base address                                 |
| <i>flag</i> | The value to be written to the interrupt flag1 register. |

**18.2.4.29** void FLEXCAN\_HAL\_GetErrCounter ( CAN\_Type \* *base*, flexcan\_buserr\_counter\_t \* *errCount* )

### Parameters

|                 |                                                  |
|-----------------|--------------------------------------------------|
| <i>base</i>     | The FlexCAN base address                         |
| <i>errCount</i> | Transmit error counter and receive error counter |

**18.2.4.30** static uint32\_t FLEXCAN\_HAL\_GetErrStatus ( CAN\_Type \* *base* ) [inline], [static]

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
|-------------|--------------------------|

### Returns

The current error and status

**18.2.4.31** void FLEXCAN\_HAL\_ClearErrIntStatusFlag ( CAN\_Type \* *base* )

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
|-------------|--------------------------|

**18.2.4.32** void FLEXCAN\_HAL\_SetRxMaskType ( CAN\_Type \* *base*, flexcan\_rx\_mask\_type\_t *type* )

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
| <i>type</i> | The FlexCAN Rx mask type |

**18.2.4.33 void FLEXCAN\_HAL\_SetRxFifoGlobalStdMask ( CAN\_Type \* *base*, uint32\_t *stdMask* )**

Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | The FlexCAN base address |
| <i>stdMask</i> | Standard mask            |

**18.2.4.34 void FLEXCAN\_HAL\_SetRxFifoGlobalExtMask ( CAN\_Type \* *base*, uint32\_t *extMask* )**

Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | The FlexCAN base address |
| <i>extMask</i> | Extended mask            |

**18.2.4.35 flexcan\_status\_t FLEXCAN\_HAL\_SetRxIndividualStdMask ( CAN\_Type \* *base*, uint32\_t *msgBuffIdx*, uint32\_t *stdMask* )**

Parameters

|                   |                             |
|-------------------|-----------------------------|
| <i>base</i>       | The FlexCAN base address    |
| <i>msgBuffIdx</i> | Index of the message buffer |
| <i>stdMask</i>    | Individual standard mask    |

Returns

0 if successful; non-zero failed

**18.2.4.36 flexcan\_status\_t FLEXCAN\_HAL\_SetRxIndividualExtMask ( CAN\_Type \* *base*, uint32\_t *msgBuffIdx*, uint32\_t *extMask* )**

## FlexCAN HAL driver

### Parameters

|                   |                             |
|-------------------|-----------------------------|
| <i>base</i>       | The FlexCAN base address    |
| <i>msgBuffIdx</i> | Index of the message buffer |
| <i>extMask</i>    | Individual extended mask    |

### Returns

0 if successful; non-zero failed

#### 18.2.4.37 void FLEXCAN\_HAL\_SetRxMsgBuffGlobalStdMask ( CAN\_Type \* *base*, uint32\_t *stdMask* )

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | The FlexCAN base address |
| <i>stdMask</i> | Standard mask            |

#### 18.2.4.38 void FLEXCAN\_HAL\_SetRxMsgBuff14StdMask ( CAN\_Type \* *base*, uint32\_t *stdMask* )

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | The FlexCAN base address |
| <i>stdMask</i> | Standard mask            |

#### 18.2.4.39 void FLEXCAN\_HAL\_SetRxMsgBuff15StdMask ( CAN\_Type \* *base*, uint32\_t *stdMask* )

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | The FlexCAN base address |
| <i>stdMask</i> | Standard mask            |

### Returns

0 if successful; non-zero failed

18.2.4.40 void FLEXCAN\_HAL\_SetRxMsgBuffGlobalExtMask ( CAN\_Type \* *base*,  
uint32\_t *extMask* )

## FlexCAN HAL driver

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | The FlexCAN base address |
| <i>extMask</i> | Extended mask            |

#### 18.2.4.41 void FLEXCAN\_HAL\_SetRxMsgBuff14ExtMask ( CAN\_Type \* *base*, uint32\_t *extMask* )

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | The FlexCAN base address |
| <i>extMask</i> | Extended mask            |

#### 18.2.4.42 void FLEXCAN\_HAL\_SetRxMsgBuff15ExtMask ( CAN\_Type \* *base*, uint32\_t *extMask* )

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | The FlexCAN base address |
| <i>extMask</i> | Extended mask            |

#### 18.2.4.43 static uint32\_t FLEXCAN\_HAL\_GetRxFifoHitIdAcceptanceFilter ( CAN\_Type \* *base* ) [inline], [static]

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | The FlexCAN base address |
|-------------|--------------------------|

### Returns

RX FIFO information



## 18.3 FlexCAN Driver

### 18.3.1 Overview

This section describes the programming interface of the FlexCAN Peripheral driver.

### 18.3.2 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.

### 18.3.3 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, 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 = kFlexCanRxFifoIDFilters_8;
flexcan1_data.is_rx_fifo_needed = true;
```

### 18.3.4 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). This list shows the CAN

## FlexCAN Driver

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  $\geq$  2. RJW is set to the minimum of 4 or to the PSEG1.

### 18.3.5 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 by calling [FLEXCAN\\_DRV\\_SetMaskType\(\)](#), [FLEXCAN\\_DRV\\_SetRxFifoGlobalMask\(\)](#), [FLEXCAN\\_DRV\\_SetRxMbGlobalMask\(\)](#) and [FLEXCAN\\_DRV\\_SetRxIndividualMask\(\)](#) functions. Then configure the RX buffers by calling either [FLEXCAN\\_DRV\\_ConfigRxMb\(\)](#) or [FLEXCAN\\_DRV\\_ConfigRxFifo\(\)](#) functions depending on the mode of reception. Finally, call [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.*
- [flexcan\\_status\\_t](#) [FLEXCAN\\_DRV\\_GetTransmitStatus](#) (uint32\_t instance)  
*Returns whether the previous FLEXCAN transmit has finished.*
- [flexcan\\_status\\_t](#) [FLEXCAN\\_DRV\\_GetReceiveStatus](#) (uint32\_t instance)  
*Returns whether the previous FLEXCAN receive is complete.*

### Variables

- CAN\_Type \*const [g\\_flexcanBase](#) []  
*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](#) []  
*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\\_SetRxMaskType](#) (uint8\_t instance, [flexcan\\_rx\\_mask\\_type\\_t](#) type)  
*Sets the RX masking type.*
- [flexcan\\_status\\_t FLEXCAN\\_DRV\\_SetRxFifoGlobalMask](#) (uint8\_t instance, [flexcan\\_msgbuff\\_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\\_msgbuff\\_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\\_msgbuff\\_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](#) (uint32\_t instance, [flexcan\\_state\\_t](#) \*state, const [flexcan\\_user\\_config\\_t](#) \*data)  
*Initializes the FlexCAN peripheral.*
- uint32\_t [FLEXCAN\\_DRV\\_Deinit](#) (uint8\_t instance)  
*Shuts down a FlexCAN instance.*

## FlexCAN Driver

### 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_SendBlocking` (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.*
- `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)  
*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_RxMessageBufferBlocking` (uint8\_t instance, uint32\_t mb\_idx, `flexcan_msgbuff_t *data`, uint32\_t timeout\_ms)  
*FlexCAN is waiting to receive data from the message buffer.*
- `flexcan_status_t FLEXCAN_DRV_RxMessageBuffer` (uint8\_t instance, uint32\_t mb\_idx, `flexcan_msgbuff_t *data`)  
*FlexCAN is waiting to receive data from the message buffer.*
- `flexcan_status_t FLEXCAN_DRV_RxFifoBlocking` (uint8\_t instance, `flexcan_msgbuff_t *data`, uint32\_t timeout\_ms)  
*FlexCAN is waiting to receive data from the message FIFO.*
- `flexcan_status_t FLEXCAN_DRV_RxFifo` (uint8\_t instance, `flexcan_msgbuff_t *data`)  
*FlexCAN is waiting to receive data from the message FIFO.*

## 18.3.6 Data Structure Documentation

### 18.3.6.1 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.

#### Data Fields

- `flexcan_msgbuff_t * fifo_message`  
*The FlexCAN receive FIFO data.*
- `flexcan_msgbuff_t * mb_message`

- *The FlexCAN receive MB data.*
- volatile uint32\_t [rx\\_mb\\_idx](#)  
*Index of the message buffer for receiving.*
- volatile uint32\_t [tx\\_mb\\_idx](#)  
*Index of the message buffer for transmitting.*
- [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.*
- 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.*

### 18.3.6.1.0.30 Field Documentation

18.3.6.1.0.30.1 [semaphore\\_t flexcan\\_state\\_t::txIrqSync](#)

18.3.6.1.0.30.2 [semaphore\\_t flexcan\\_state\\_t::rxIrqSync](#)

18.3.6.1.0.30.3 [volatile bool flexcan\\_state\\_t::isTxBusy](#)

18.3.6.1.0.30.4 [volatile bool flexcan\\_state\\_t::isRxBusy](#)

18.3.6.1.0.30.5 [volatile bool flexcan\\_state\\_t::isTxBlocking](#)

18.3.6.1.0.30.6 [volatile bool flexcan\\_state\\_t::isRxBlocking](#)

### 18.3.6.2 struct flexcan\_data\_info\_t

#### Data Fields

- [flexcan\\_msgbuff\\_id\\_type\\_t msg\\_id\\_type](#)  
*Type of message ID (standard or extended)*
- uint32\_t [data\\_length](#)  
*Length of Data in Bytes.*

### 18.3.6.3 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](#)  
*The number of RX FIFO ID filters needed.*
- bool [is\\_rx\\_fifo\\_needed](#)

## FlexCAN Driver

- 1 if needed; 0 if not.*
- `flexcan_operation_modes_t flexcanMode`  
*User configurable FlexCAN operation modes.*

### 18.3.6.3.0.31 Field Documentation

#### 18.3.6.3.0.31.1 `bool flexcan_user_config_t::is_rx_fifo_needed`

This controls whether the Rx FIFO feature is enabled or not.

## 18.3.7 Function Documentation

### 18.3.7.1 `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

### 18.3.7.2 `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

### 18.3.7.3 `void FLEXCAN_DRV_SetRxMaskType ( 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  |

**18.3.7.4 flexcan\_status\_t FLEXCAN\_DRV\_SetRxFifoGlobalMask ( uint8\_t *instance*, flexcan\_msgbuff\_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

**18.3.7.5 flexcan\_status\_t FLEXCAN\_DRV\_SetRxMbGlobalMask ( uint8\_t *instance*, flexcan\_msgbuff\_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

**18.3.7.6 flexcan\_status\_t FLEXCAN\_DRV\_SetRxIndividualMask ( uint8\_t *instance*, flexcan\_msgbuff\_id\_type\_t *id\_type*, uint32\_t *mb\_idx*, uint32\_t *mask* )**

## FlexCAN Driver

### Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <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.

### 18.3.7.7 flexcan\_status\_t FLEXCAN\_DRV\_Init ( uint32\_t *instance*, flexcan\_state\_t \* *state*, const flexcan\_user\_config\_t \* *data* )

This function initializes

### Parameters

|                 |                                                |
|-----------------|------------------------------------------------|
| <i>instance</i> | A FlexCAN instance number                      |
| <i>state</i>    | Pointer to the FlexCAN driver state structure. |
| <i>data</i>     | The FlexCAN platform data                      |

### Returns

0 if successful; non-zero failed

### 18.3.7.8 uint32\_t FLEXCAN\_DRV\_Deinit ( uint8\_t *instance* )

### Parameters

|                 |                           |
|-----------------|---------------------------|
| <i>instance</i> | A FlexCAN instance number |
|-----------------|---------------------------|

### Returns

0 if successful; non-zero failed

### 18.3.7.9 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

**18.3.7.10 flexcan\_status\_t FLEXCAN\_DRV\_SendBlocking ( 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 milliseconds. |

Returns

0 if successful; non-zero failed

**18.3.7.11 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* )**

Parameters

---

## FlexCAN Driver

|                 |                               |
|-----------------|-------------------------------|
| <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. |

Returns

0 if successful; non-zero failed

### 18.3.7.12 flexcan\_status\_t FLEXCAN\_DRV\_ConfigRxMb ( uint8\_t *instance*, uint32\_t *mb\_idx*, flexcan\_data\_info\_t \* *rx\_info*, uint32\_t *msg\_id* )

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <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

### 18.3.7.13 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.

**18.3.7.14 flexcan\_status\_t FLEXCAN\_DRV\_RxMessageBufferBlocking ( uint8\_t *instance*, uint32\_t *mb\_idx*, flexcan\_msgbuff\_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 milliseconds. |

Returns

0 if successful; non-zero failed

**18.3.7.15 flexcan\_status\_t FLEXCAN\_DRV\_RxMessageBuffer ( uint8\_t *instance*, uint32\_t *mb\_idx*, flexcan\_msgbuff\_t \* *data* )**

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. |

Returns

0 if successful; non-zero failed

**18.3.7.16 flexcan\_status\_t FLEXCAN\_DRV\_RxFifoBlocking ( uint8\_t *instance*, flexcan\_msgbuff\_t \* *data*, uint32\_t *timeout\_ms* )**

## FlexCAN Driver

### 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 milliseconds. |

### Returns

0 if successful; non-zero failed

### 18.3.7.17 flexcan\_status\_t FLEXCAN\_DRV\_RxFifo ( uint8\_t instance, flexcan\_msgbuff\_t \* data )

### Parameters

|                 |                                          |
|-----------------|------------------------------------------|
| <i>instance</i> | A FlexCAN instance number                |
| <i>data</i>     | The FlexCAN receive message buffer data. |

### Returns

0 if successful; non-zero failed

### 18.3.7.18 void FLEXCAN\_DRV\_IRQHandler ( uint8\_t instance )

### Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>instance</i> | The FlexCAN instance number. |
|-----------------|------------------------------|

### 18.3.7.19 flexcan\_status\_t FLEXCAN\_DRV\_GetTransmitStatus ( uint32\_t instance )

When performing an async transmit, call this function to ascertain the state of the current transmission: in progress (or busy) or complete (success).

### Parameters

---

|                 |                                  |
|-----------------|----------------------------------|
| <i>instance</i> | The FLEXCAN module base address. |
|-----------------|----------------------------------|

Returns

The transmit status.

Return values

|                                |                                          |
|--------------------------------|------------------------------------------|
| <i>kStatus_FLEXCAN_Success</i> | The transmit has completed successfully. |
| <i>kStatus_FLEXCAN_Tx-Busy</i> | The transmit is still in progress.       |

### 18.3.7.20 flexcan\_status\_t FLEXCAN\_DRV\_GetReceiveStatus ( uint32\_t instance )

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).

Parameters

|                        |                                                                                                                         |
|------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>        | The FLEXCAN module base address.                                                                                        |
| <i>bytes-Remaining</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_FLEXCAN_Success</i> | The receive has completed successfully. |
| <i>kStatus_FLEXCAN_Rx-Busy</i> | The receive is still in progress.       |

## FlexCAN Driver

### 18.3.8 Variable Documentation

18.3.8.1 `CAN_Type* const g_flexcanBase[]`

18.3.8.2 `const IRQn_Type g_flexcanRxWarningIrqId[]`

18.3.8.3 `const IRQn_Type g_flexcanTxWarningIrqId[]`

18.3.8.4 `const IRQn_Type g_flexcanWakeUpIrqId[]`

18.3.8.5 `const IRQn_Type g_flexcanErrorIrqId[]`

18.3.8.6 `const IRQn_Type g_flexcanBusOffIrqId[]`

18.3.8.7 `const IRQn_Type g_flexcanOredMessageBufferIrqId[]`

## Chapter 19

### FlexTimer (FTM)

#### 19.1 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

### 19.2 FlexTimer HAL driver

#### 19.2.1 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\\_dual\\_edge\\_capture\\_param\\_t](#)  
*FlexTimer Dual Edge Capture parameters. [More...](#)*
- struct [ftm\\_phase\\_params\\_t](#)  
*FlexTimer quadrature decode phase parameters. [More...](#)*

#### Macros

- #define [CHAN0\\_IDX](#) (0U)  
*Channel number for CHAN0.*
- #define [CHAN1\\_IDX](#) (1U)  
*Channel number for CHAN1.*
- #define [CHAN2\\_IDX](#) (2U)  
*Channel number for CHAN2.*
- #define [CHAN3\\_IDX](#) (3U)  
*Channel number for CHAN3.*
- #define [CHAN4\\_IDX](#) (4U)  
*Channel number for CHAN4.*
- #define [CHAN5\\_IDX](#) (5U)  
*Channel number for CHAN5.*
- #define [CHAN6\\_IDX](#) (6U)  
*Channel number for CHAN6.*
- #define [CHAN7\\_IDX](#) (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` {  
`kFtmQuadPhaseNormal` = 0,  
`kFtmQuadPhaseInvert` }
  - 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` {  
`kFtmBdmMode_00` = 0,  
`kFtmBdmMode_01`,  
`kFtmBdmMode_10`,  
`kFtmBdmMode_11` }
  - Options for the FlexTimer behaviour in BDM Mode.*
  - enum `ftm_status_t` {  
`kStatusFtmSuccess` = 0U,  
`kStatusFtmError` = 1U }
  - FTM status.*

## Functions

- static void `FTM_HAL_SetClockSource` (FTM\_Type \*ftmBase, `ftm_clock_source_t` clock)  
*Sets the FTM clock source.*
- static uint8\_t `FTM_HAL_GetClockSource` (FTM\_Type \*ftmBase)  
*Reads the FTM clock source.*
- static void `FTM_HAL_SetClockPs` (FTM\_Type \*ftmBase, `ftm_clock_ps_t` ps)  
*Sets the FTM clock divider.*
- static uint8\_t `FTM_HAL_GetClockPs` (FTM\_Type \*ftmBase)  
*Reads the FTM clock divider.*
- static void `FTM_HAL_EnableTimerOverflowInt` (FTM\_Type \*ftmBase)  
*Enables the FTM peripheral timer overflow interrupt.*
- static void `FTM_HAL_DisableTimerOverflowInt` (FTM\_Type \*ftmBase)  
*Disables the FTM peripheral timer overflow interrupt.*
- static bool `FTM_HAL_IsOverflowIntEnabled` (FTM\_Type \*ftmBase)  
*Reads the bit that controls enabling the FTM timer overflow interrupt.*
- static void `FTM_HAL_ClearTimerOverflow` (FTM\_Type \*ftmBase)  
*Clears the timer overflow interrupt flag.*
- static bool `FTM_HAL_HasTimerOverflowed` (FTM\_Type \*ftmBase)  
*Returns the FTM peripheral timer overflow interrupt flag.*
- static void `FTM_HAL_SetCpwms` (FTM\_Type \*ftmBase, uint8\_t mode)

## FlexTimer HAL driver

- Sets the FTM center-aligned PWM select.*

  - static void [FTM\\_HAL\\_SetCounter](#) (FTM\_Type \*ftmBase, uint16\_t val)
- Sets the FTM peripheral current counter value.*

  - static uint16\_t [FTM\\_HAL\\_GetCounter](#) (FTM\_Type \*ftmBase)
- Returns the FTM peripheral current counter value.*

  - static void [FTM\\_HAL\\_SetMod](#) (FTM\_Type \*ftmBase, uint16\_t val)
- Sets the FTM peripheral timer modulo value.*

  - static uint16\_t [FTM\\_HAL\\_GetMod](#) (FTM\_Type \*ftmBase)
- Returns the FTM peripheral counter modulo value.*

  - static void [FTM\\_HAL\\_SetCounterInitVal](#) (FTM\_Type \*ftmBase, uint16\_t val)
- Sets the FTM peripheral timer counter initial value.*

  - static uint16\_t [FTM\\_HAL\\_GetCounterInitVal](#) (FTM\_Type \*ftmBase)
- Returns the FTM peripheral counter initial value.*

  - static void [FTM\\_HAL\\_SetChnMSnBAMode](#) (FTM\_Type \*ftmBase, uint8\_t channel, uint8\_t selection)
- Sets the FTM peripheral timer channel mode.*

  - static void [FTM\\_HAL\\_SetChnEdgeLevel](#) (FTM\_Type \*ftmBase, uint8\_t channel, uint8\_t level)
- Sets the FTM peripheral timer channel edge level.*

  - static uint8\_t [FTM\\_HAL\\_GetChnMode](#) (FTM\_Type \*ftmBase, uint8\_t channel)
- Gets the FTM peripheral timer channel mode.*

  - static uint8\_t [FTM\\_HAL\\_GetChnEdgeLevel](#) (FTM\_Type \*ftmBase, uint8\_t channel)
- Gets the FTM peripheral timer channel edge level.*

  - static void [FTM\\_HAL\\_SetChnDmaCmd](#) (FTM\_Type \*ftmBase, uint8\_t channel, bool val)
- Enables or disables the FTM peripheral timer channel DMA.*

  - static bool [FTM\\_HAL\\_IsChnDma](#) (FTM\_Type \*ftmBase, uint8\_t channel)
- Returns whether the FTM peripheral timer channel DMA is enabled.*

  - static bool [FTM\\_HAL\\_IsChnIntEnabled](#) (FTM\_Type \*ftmBase, uint8\_t channel)
- Get FTM channel(n) interrupt enabled or not.*

  - static void [FTM\\_HAL\\_EnableChnInt](#) (FTM\_Type \*ftmBase, uint8\_t channel)
- Enables the FTM peripheral timer channel(n) interrupt.*

  - static void [FTM\\_HAL\\_DisableChnInt](#) (FTM\_Type \*ftmBase, uint8\_t channel)
- Disables the FTM peripheral timer channel(n) interrupt.*

  - static bool [FTM\\_HAL\\_HasChnEventOccurred](#) (FTM\_Type \*ftmBase, uint8\_t channel)
- Returns whether any event for the FTM peripheral timer channel has occurred.*

  - static void [FTM\\_HAL\\_ClearChnEventFlag](#) (FTM\_Type \*ftmBase, uint8\_t channel)
- Clear the channel flag by writing a 0 to the CHF bit.*

  - static void [FTM\\_HAL\\_SetChnCountVal](#) (FTM\_Type \*ftmBase, uint8\_t channel, uint16\_t val)
- Sets the FTM peripheral timer channel counter value.*

  - static uint16\_t [FTM\\_HAL\\_GetChnCountVal](#) (FTM\_Type \*ftmBase, uint8\_t channel)
- Gets the FTM peripheral timer channel counter value.*

  - static uint32\_t [FTM\\_HAL\\_GetChnEventStatus](#) (FTM\_Type \*ftmBase, uint8\_t channel)
- Gets the FTM peripheral timer channel event status.*

  - static void [FTM\\_HAL\\_ClearChnEventStatus](#) (FTM\_Type \*ftmBase, uint8\_t channel)
- Clears the FTM peripheral timer all channel event status.*

  - static void [FTM\\_HAL\\_SetOutmaskReg](#) (FTM\_Type \*ftmBase, uint32\_t regVal)
- Writes the provided value to the OUTMASK register.*

  - static void [FTM\\_HAL\\_SetChnOutputMask](#) (FTM\_Type \*ftmBase, uint8\_t channel, bool mask)
- Sets the FTM peripheral timer channel output mask.*

  - static void [FTM\\_HAL\\_SetChnOutputInitStateCmd](#) (FTM\_Type \*ftmBase, uint8\_t channel, uint8\_t state)
- Sets the FTM peripheral timer channel output initial state 0 or 1.*

- static void [FTM\\_HAL\\_SetChnOutputPolarityCmd](#) (FTM\_Type \*ftmBase, uint8\_t channel, uint8\_t pol)  
*Sets the FTM peripheral timer channel output polarity.*
- static void [FTM\\_HAL\\_SetChnFaultInputPolarityCmd](#) (FTM\_Type \*ftmBase, uint8\_t channel, uint8\_t pol)  
*Sets the FTM peripheral timer channel input polarity.*
- static void [FTM\\_HAL\\_EnableFaultInt](#) (FTM\_Type \*ftmBase)  
*Enables the FTM peripheral timer fault interrupt.*
- static void [FTM\\_HAL\\_DisableFaultInt](#) (FTM\_Type \*ftmBase)  
*Disables the FTM peripheral timer fault interrupt.*
- static void [FTM\\_HAL\\_SetFaultControlMode](#) (FTM\_Type \*ftmBase, uint8\_t mode)  
*Defines the FTM fault control mode.*
- static void [FTM\\_HAL\\_SetCaptureTestCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Enables or disables the FTM peripheral timer capture test mode.*
- static void [FTM\\_HAL\\_SetWriteProtectionCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Enables or disables the FTM write protection.*
- static void [FTM\\_HAL\\_Enable](#) (FTM\_Type \*ftmBase, bool enable)  
*Enables the FTM peripheral timer group.*
- static void [FTM\\_HAL\\_SetInitChnOutputCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Initializes the channels output.*
- static void [FTM\\_HAL\\_SetPwmSyncMode](#) (FTM\_Type \*ftmBase, bool enable)  
*Sets the FTM peripheral timer sync mode.*
- static void [FTM\\_HAL\\_SetSoftwareTriggerCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Enables or disables the FTM peripheral timer software trigger.*
- void [FTM\\_HAL\\_SetHardwareSyncTriggerSrc](#) (FTM\_Type \*ftmBase, uint32\_t trigger\_num, bool enable)  
*Sets the FTM peripheral timer hardware trigger.*
- static void [FTM\\_HAL\\_SetOutmaskPwmSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Determines when the OUTMASK register is updated with the value of its buffer.*
- static void [FTM\\_HAL\\_SetCountReinitSyncCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Determines if the FTM counter is re-initialized when the selected trigger for synchronization is detected.*
- static void [FTM\\_HAL\\_SetMaxLoadingCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Enables or disables the FTM peripheral timer maximum loading points.*
- static void [FTM\\_HAL\\_SetMinLoadingCmd](#) (FTM\_Type \*ftmBase, 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](#) (FTM\_Type \*ftmBase, uint8\_t chnlPairNum, bool enable)  
*Enables the FTM peripheral timer channel pair fault control.*
- static void [FTM\\_HAL\\_SetDualChnPwmSyncCmd](#) (FTM\_Type \*ftmBase, uint8\_t chnlPairNum, bool enable)  
*Enables or disables the FTM peripheral timer channel pair counter PWM sync.*
- static void [FTM\\_HAL\\_SetDualChnDeadtimeCmd](#) (FTM\_Type \*ftmBase, uint8\_t chnlPairNum, bool enable)  
*Enables or disabled the FTM peripheral timer channel pair deadtime insertion.*
- static void [FTM\\_HAL\\_SetDualChnDecapCmd](#) (FTM\_Type \*ftmBase, uint8\_t chnlPairNum, bool enable)  
*Enables or disables the FTM peripheral timer channel dual edge capture decap.*
- static void [FTM\\_HAL\\_SetDualEdgeCaptureCmd](#) (FTM\_Type \*ftmBase, uint8\_t chnlPairNum,

## FlexTimer HAL driver

- bool enable)
  - Enables the FTM peripheral timer dual edge capture mode.*
- static void [FTM\\_HAL\\_SetDualChnCompCmd](#) (FTM\_Type \*ftmBase, uint8\_t chnlPairNum, bool enable)
  - Enables or disables the FTM peripheral timer channel pair output complement mode.*
- static void [FTM\\_HAL\\_SetDualChnCombineCmd](#) (FTM\_Type \*ftmBase, uint8\_t chnlPairNum, bool enable)
  - Enables or disables the FTM peripheral timer channel pair output combine mode.*
- static void [FTM\\_HAL\\_SetDeadtimePrescale](#) (FTM\_Type \*ftmBase, [ftm\\_deadtime\\_ps\\_t](#) divider)
  - Sets the FTM deadtime divider.*
- static void [FTM\\_HAL\\_SetDeadtimeCount](#) (FTM\_Type \*ftmBase, uint8\_t count)
  - Sets the FTM deadtime value.*
- static void [FTM\\_HAL\\_SetInitTriggerCmd](#) (FTM\_Type \*ftmBase, bool enable)
  - Enables or disables the generation of the trigger when the FTM counter is equal to the CNTIN register.*
- void [FTM\\_HAL\\_SetChnTriggerCmd](#) (FTM\_Type \*ftmBase, uint8\_t channel, bool val)
  - Enables or disables the generation of the FTM peripheral timer channel trigger.*
- static bool [FTM\\_HAL\\_IsChnTriggerGenerated](#) (FTM\_Type \*ftmBase)
  - Checks whether any channel trigger event has occurred.*
- static uint8\_t [FTM\\_HAL\\_GetDetectedFaultInput](#) (FTM\_Type \*ftmBase)
  - Gets the FTM detected fault input.*
- static bool [FTM\\_HAL\\_IsWriteProtectionEnabled](#) (FTM\_Type \*ftmBase)
  - Checks whether the write protection is enabled.*
- static void [FTM\\_HAL\\_SetQuadDecoderCmd](#) (FTM\_Type \*ftmBase, bool enable)
  - Enables the channel quadrature decoder.*
- static void [FTM\\_HAL\\_SetQuadPhaseAFilterCmd](#) (FTM\_Type \*ftmBase, bool enable)
  - Enables or disables the phase A input filter.*
- static void [FTM\\_HAL\\_SetQuadPhaseBFilterCmd](#) (FTM\_Type \*ftmBase, bool enable)
  - Enables or disables the phase B input filter.*
- static void [FTM\\_HAL\\_SetQuadPhaseAPolarity](#) (FTM\_Type \*ftmBase, [ftm\\_quad\\_phase\\_polarity\\_t](#) mode)
  - Selects polarity for the quadrature decode phase A input.*
- static void [FTM\\_HAL\\_SetQuadPhaseBPolarity](#) (FTM\_Type \*ftmBase, [ftm\\_quad\\_phase\\_polarity\\_t](#) mode)
  - Selects polarity for the quadrature decode phase B input.*
- static void [FTM\\_HAL\\_SetQuadMode](#) (FTM\_Type \*ftmBase, [ftm\\_quad\\_decode\\_mode\\_t](#) quad-Mode)
  - Sets the encoding mode used in quadrature decoding mode.*
- static uint8\_t [FTM\\_HAL\\_GetQuadDir](#) (FTM\_Type \*ftmBase)
  - Gets the FTM counter direction in quadrature mode.*
- static uint8\_t [FTM\\_HAL\\_GetQuadTimerOverflowDir](#) (FTM\_Type \*ftmBase)
  - Gets the Timer overflow direction in quadrature mode.*
- void [FTM\\_HAL\\_SetChnInputCaptureFilter](#) (FTM\_Type \*ftmBase, uint8\_t channel, uint8\_t val)
  - Sets the FTM peripheral timer channel input capture filter value.*
- static void [FTM\\_HAL\\_SetFaultInputFilterVal](#) (FTM\_Type \*ftmBase, uint32\_t val)
  - Sets the fault input filter value.*
- static void [FTM\\_HAL\\_SetFaultInputFilterCmd](#) (FTM\_Type \*ftmBase, uint8\_t inputNum, bool val)
  - Enables or disables the fault input filter.*
- static void [FTM\\_HAL\\_SetFaultInputCmd](#) (FTM\_Type \*ftmBase, uint8\_t inputNum, bool val)
  - Enables or disables the fault input.*
- static void [FTM\\_HAL\\_SetDualChnInvertCmd](#) (FTM\_Type \*ftmBase, uint8\_t chnlPairNum, bool

val)

*Enables or disables the channel invert for a channel pair.*

- static void [FTM\\_HAL\\_SetInvctrlReg](#) (FTM\_Type \*ftmBase, uint32\_t regVal)  
*Writes the provided value to the Inverting control register.*
- static void [FTM\\_HAL\\_SetChnSoftwareCtrlCmd](#) (FTM\_Type \*ftmBase, uint8\_t channel, bool val)  
*Enables or disables the channel software output control.*
- static void [FTM\\_HAL\\_SetChnSoftwareCtrlVal](#) (FTM\_Type \*ftmBase, uint8\_t channel, bool val)  
*Sets the channel software output control value.*
- static void [FTM\\_HAL\\_SetPwmLoadCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Enables or disables the loading of MOD, CNTIN and CV with values of their write buffer.*
- static void [FTM\\_HAL\\_SetPwmLoadChnSelCmd](#) (FTM\_Type \*ftmBase, uint8\_t channel, bool val)  
*Includes or excludes the channel in the matching process.*
- static void [FTM\\_HAL\\_SetGlobalTimeBaseOutputCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Enables or disables the FTM global time base signal generation to other FTM's.*
- static void [FTM\\_HAL\\_SetGlobalTimeBaseCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Enables or disables the FTM timer global time base.*
- static void [FTM\\_HAL\\_SetBdmMode](#) (FTM\_Type \*ftmBase, [ftm\\_bdm\\_mode\\_t](#) val)  
*Sets the BDM mode.*
- static void [FTM\\_HAL\\_SetTofFreq](#) (FTM\_Type \*ftmBase, uint8\_t val)  
*Sets the FTM timer TOF Frequency.*
- void [FTM\\_HAL\\_SetSyncMode](#) (FTM\_Type \*ftmBase, uint32\_t syncMethod)  
*Sets the FTM register synchronization method.*
- static void [FTM\\_HAL\\_SetSwoctrlHardwareSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Sets the sync mode for the FTM SWOCTRL register when using a hardware trigger.*
- static void [FTM\\_HAL\\_SetInvctrlHardwareSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Sets sync mode for FTM INVCTRL register when using a hardware trigger.*
- static void [FTM\\_HAL\\_SetOutmaskHardwareSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Sets sync mode for FTM OUTMASK register when using a hardware trigger.*
- static void [FTM\\_HAL\\_SetModCntrlCvHardwareSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Sets sync mode for FTM MOD, CNTIN and CV registers when using a hardware trigger.*
- static void [FTM\\_HAL\\_SetCounterHardwareSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Sets sync mode for FTM counter register when using a hardware trigger.*
- static void [FTM\\_HAL\\_SetSwoctrlSoftwareSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Sets sync mode for FTM SWOCTRL register when using a software trigger.*
- static void [FTM\\_HAL\\_SetInvctrlSoftwareSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Sets sync mode for FTM INVCTRL register when using a software trigger.*
- static void [FTM\\_HAL\\_SetOutmaskSoftwareSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Sets sync mode for FTM OUTMASK register when using a software trigger.*
- static void [FTM\\_HAL\\_SetModCntrlCvSoftwareSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Sets synch mode for FTM MOD, CNTIN and CV registers when using a software trigger.*
- static void [FTM\\_HAL\\_SetCounterSoftwareSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Sets sync mode for FTM counter register when using a software trigger.*
- static void [FTM\\_HAL\\_SetPwmSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Sets the PWM synchronization mode to enhanced or legacy.*
- static void [FTM\\_HAL\\_SetSwoctrlPwmSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Sets the SWOCTRL register PWM synchronization mode.*
- static void [FTM\\_HAL\\_SetInvctrlPwmSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)  
*Sets the INVCTRL register PWM synchronization mode.*
- static void [FTM\\_HAL\\_SetCntrlPwmSyncModeCmd](#) (FTM\_Type \*ftmBase, bool enable)

## FlexTimer HAL driver

- *Sets the CNTIN register PWM synchronization mode.*
- void [FTM\\_HAL\\_Reset](#) (FTM\_Type \*ftmBase)
- *Resets the FTM registers.*
- void [FTM\\_HAL\\_Init](#) (FTM\_Type \*ftmBase)
- *Initializes the FTM.*
- void [FTM\\_HAL\\_EnablePwmMode](#) (FTM\_Type \*ftmBase, [ftm\\_pwm\\_param\\_t](#) \*config, uint8\_t channel)
- *Enables the FTM timer when it is PWM output mode.*
- void [FTM\\_HAL\\_DisablePwmMode](#) (FTM\_Type \*ftmBase, [ftm\\_pwm\\_param\\_t](#) \*config, uint8\_t channel)
- *Disables the PWM output mode.*

## 19.2.2 Data Structure Documentation

### 19.2.2.1 union [ftm\\_edge\\_mode\\_t](#)

### 19.2.2.2 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.*

#### 19.2.2.2.0.32 Field Documentation

##### 19.2.2.2.0.32.1 uint32\_t [ftm\\_pwm\\_param\\_t::uDutyCyclePercent](#)

100=active signal (100% duty cycle).

##### 19.2.2.2.0.32.2 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

### 19.2.2.3 struct [ftm\\_dual\\_edge\\_capture\\_param\\_t](#)

#### Data Fields

- [ftm\\_dual\\_capture\\_edge\\_mode\\_t mode](#)

- *Dual Edge Capture mode: one-shot or continuous.*
- `ftm_input_capture_edge_mode_t currChanEdgeMode`  
*Input Edge select for Channel n.*
- `ftm_input_capture_edge_mode_t nextChanEdgeMode`  
*Input Edge select for Channel n + 1.*

#### 19.2.2.4 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*

#### 19.2.3 Macro Definition Documentation

19.2.3.1 `#define CHAN0_IDX (0U)`

19.2.3.2 `#define CHAN1_IDX (1U)`

19.2.3.3 `#define CHAN2_IDX (2U)`

19.2.3.4 `#define CHAN3_IDX (3U)`

19.2.3.5 `#define CHAN4_IDX (4U)`

19.2.3.6 `#define CHAN5_IDX (5U)`

19.2.3.7 `#define CHAN6_IDX (6U)`

19.2.3.8 `#define CHAN7_IDX (7U)`

#### 19.2.4 Enumeration Type Documentation

19.2.4.1 enum `ftm_output_compare_edge_mode_t`

Toggle, clear or set.

## FlexTimer HAL driver

### 19.2.4.2 enum `ftm_quad_phase_polarity_t`

Enumerator

***kFtmQuadPhaseNormal*** Phase A input signal is not inverted before identifying the rising and falling edges of this signal.

***kFtmQuadPhaseInvert*** Phase A input signal is inverted before identifying the rising and falling edges of this signal.

### 19.2.4.3 enum `ftm_bdm_mode_t`

Enumerator

***kFtmBdmMode\_00*** FTM counter stopped, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers bypass the register buffers.

***kFtmBdmMode\_01*** FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are forced to their safe value , writes to MOD,CNTIN and C(n)V registers bypass the register buffers.

***kFtmBdmMode\_10*** FTM counter stopped, CH(n)F bit is not set, FTM channels outputs are frozen when chip enters in BDM mode, writes to MOD,CNTIN and C(n)V registers bypass the register buffers.

***kFtmBdmMode\_11*** FTM counter in functional mode, CH(n)F bit can be set, FTM channels in functional mode, writes to MOD,CNTIN and C(n)V registers is in fully functional mode.

### 19.2.4.4 enum `ftm_status_t`

Enumerator

***kStatusFtmSuccess*** FTM success status.

***kStatusFtmError*** FTM error status.

## 19.2.5 Function Documentation

**19.2.5.1 static void FTM\_HAL\_SetClockSource ( FTM\_Type \* *ftmBase*, *ftm\_clock\_source\_t clock* ) [inline], [static]**

Parameters

---



|                |                                                                                                               |
|----------------|---------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                  |
| <i>clock</i>   | The FTM peripheral clock selection<br>bits - 00: No clock 01: system clock 10: fixed clock 11: External clock |

**19.2.5.2 static uint8\_t FTM\_HAL\_GetClockSource ( FTM\_Type \* *ftmBase* ) [inline], [static]**

Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

Returns

The FTM clock source selection  
bits - 00: No clock 01: system clock 10: fixed clock 11:External clock

**19.2.5.3 static void FTM\_HAL\_SetClockPs ( FTM\_Type \* *ftmBase*, *ftm\_clock\_ps\_t ps* ) [inline], [static]**

Parameters

|                |                                            |
|----------------|--------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer               |
| <i>ps</i>      | The FTM peripheral clock pre-scale divider |

**19.2.5.4 static uint8\_t FTM\_HAL\_GetClockPs ( FTM\_Type \* *ftmBase* ) [inline], [static]**

Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

Returns

The FTM clock pre-scale divider

**19.2.5.5 static void FTM\_HAL\_EnableTimerOverflowInt ( FTM\_Type \* *ftmBase* ) [inline], [static]**

## FlexTimer HAL driver

Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

**19.2.5.6 static void FTM\_HAL\_DisableTimerOverflowInt ( FTM\_Type \* *ftmBase* )  
[inline], [static]**

Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

**19.2.5.7 static bool FTM\_HAL\_IsOverflowIntEnabled ( FTM\_Type \* *ftmBase* )  
[inline], [static]**

Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

Returns

true if overflow interrupt is enabled, false if not

**19.2.5.8 static void FTM\_HAL\_ClearTimerOverflow ( FTM\_Type \* *ftmBase* ) [inline],  
[static]**

Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

**19.2.5.9 static bool FTM\_HAL\_HasTimerOverflowed ( FTM\_Type \* *ftmBase* ) [inline],  
[static]**

Parameters

---

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

Returns

true if overflow, false if not

**19.2.5.10 static void FTM\_HAL\_SetCpwms ( FTM\_Type \* *ftmBase*, uint8\_t *mode* )  
[inline], [static]**

Parameters

|                |                                           |
|----------------|-------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer              |
| <i>mode</i>    | 1:upcounting mode 0:up_down counting mode |

**19.2.5.11 static void FTM\_HAL\_SetCounter ( FTM\_Type \* *ftmBase*, uint16\_t *val* )  
[inline], [static]**

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer      |
| <i>val</i>     | FTM timer counter value to be set |

**19.2.5.12 static uint16\_t FTM\_HAL\_GetCounter ( FTM\_Type \* *ftmBase* ) [inline],  
[static]**

Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

Returns

current FTM timer counter value

**19.2.5.13 static void FTM\_HAL\_SetMod ( FTM\_Type \* *ftmBase*, uint16\_t *val* )  
[inline], [static]**

## FlexTimer HAL driver

### Parameters

|                |                                         |
|----------------|-----------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer            |
| <i>val</i>     | The value to be set to the timer modulo |

**19.2.5.14** `static uint16_t FTM_HAL_GetMod ( FTM_Type * ftmBase ) [inline], [static]`

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

### Returns

FTM timer modulo value

**19.2.5.15** `static void FTM_HAL_SetCounterInitVal ( FTM_Type * ftmBase, uint16_t val ) [inline], [static]`

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
| <i>val</i>     | initial value to be set      |

**19.2.5.16** `static uint16_t FTM_HAL_GetCounterInitVal ( FTM_Type * ftmBase ) [inline], [static]`

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

### Returns

FTM timer counter initial value

**19.2.5.17** `static void FTM_HAL_SetChnMSnBAMode ( FTM_Type * ftmBase, uint8_t channel, uint8_t selection ) [inline], [static]`

## Parameters

|                  |                                                         |
|------------------|---------------------------------------------------------|
| <i>ftmBase</i>   | The FTM base address pointer                            |
| <i>channel</i>   | The FTM peripheral channel number                       |
| <i>selection</i> | The mode to be set valid value MSnB:MSnA :00,01, 10, 11 |

**19.2.5.18** `static void FTM_HAL_SetChnEdgeLevel ( FTM_Type * ftmBase, uint8_t channel, uint8_t level ) [inline], [static]`

## Parameters

|                |                                                                              |
|----------------|------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                 |
| <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 |

**19.2.5.19** `static uint8_t FTM_HAL_GetChnMode ( FTM_Type * ftmBase, uint8_t channel ) [inline], [static]`

## Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer      |
| <i>channel</i> | The FTM peripheral channel number |

## Returns

The MSnB:MSnA mode value, will be 00,01, 10, 11

**19.2.5.20** `static uint8_t FTM_HAL_GetChnEdgeLevel ( FTM_Type * ftmBase, uint8_t channel ) [inline], [static]`

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

## FlexTimer HAL driver

|                |                                   |
|----------------|-----------------------------------|
| <i>channel</i> | The FTM peripheral channel number |
|----------------|-----------------------------------|

Returns

The ELSnB:ELSnA mode value, will be 00,01, 10, 11

**19.2.5.21 static void FTM\_HAL\_SetChnDmaCmd ( FTM\_Type \* *ftmBase*, uint8\_t *channel*, bool *val* ) [inline], [static]**

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer      |
| <i>channel</i> | The FTM peripheral channel number |
| <i>val</i>     | enable or disable                 |

**19.2.5.22 static bool FTM\_HAL\_IsChnDma ( FTM\_Type \* *ftmBase*, uint8\_t *channel* ) [inline], [static]**

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer      |
| <i>channel</i> | The FTM peripheral channel number |

Returns

true if enabled, false if disabled

**19.2.5.23 static bool FTM\_HAL\_IsChnIntEnabled ( FTM\_Type \* *ftmBase*, uint8\_t *channel* ) [inline], [static]**

Parameters

|                |                          |
|----------------|--------------------------|
| <i>ftmBase</i> | FTM module base address. |
|----------------|--------------------------|

|                |                                   |
|----------------|-----------------------------------|
| <i>channel</i> | The FTM peripheral channel number |
|----------------|-----------------------------------|

**19.2.5.24** `static void FTM_HAL_EnableChnInt ( FTM_Type * ftmBase, uint8_t channel ) [inline], [static]`

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer      |
| <i>channel</i> | The FTM peripheral channel number |

**19.2.5.25** `static void FTM_HAL_DisableChnInt ( FTM_Type * ftmBase, uint8_t channel ) [inline], [static]`

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer      |
| <i>channel</i> | The FTM peripheral channel number |

**19.2.5.26** `static bool FTM_HAL_HasChnEventOccurred ( FTM_Type * ftmBase, uint8_t channel ) [inline], [static]`

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer      |
| <i>channel</i> | The FTM peripheral channel number |

Returns

true if event occurred, false otherwise.

**19.2.5.27** `static void FTM_HAL_ClearChnEventFlag ( FTM_Type * ftmBase, uint8_t channel ) [inline], [static]`

## FlexTimer HAL driver

### Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer      |
| <i>channel</i> | The FTM peripheral channel number |

**19.2.5.28** `static void FTM_HAL_SetChnCountVal ( FTM_Type * ftmBase, uint8_t channel, uint16_t val ) [inline], [static]`

### Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer      |
| <i>channel</i> | The FTM peripheral channel number |
| <i>val</i>     | counter value to be set           |

**19.2.5.29** `static uint16_t FTM_HAL_GetChnCountVal ( FTM_Type * ftmBase, uint8_t channel ) [inline], [static]`

### Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer      |
| <i>channel</i> | The FTM peripheral channel number |

### Returns

return channel counter value

**19.2.5.30** `static uint32_t FTM_HAL_GetChnEventStatus ( FTM_Type * ftmBase, uint8_t channel ) [inline], [static]`

### Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer      |
| <i>channel</i> | The FTM peripheral channel number |

### Returns

return channel event status value



19.2.5.31 `static void FTM_HAL_ClearChnEventStatus ( FTM_Type * ftmBase, uint8_t channel ) [inline], [static]`

## FlexTimer HAL driver

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer      |
| <i>channel</i> | The FTM peripheral channel number |

**19.2.5.32** `static void FTM_HAL_SetOutmaskReg ( FTM_Type * ftmBase, uint32_t regVal ) [inline], [static]`

This function will mask/uumask multiple channels.

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer        |
| <i>regVal</i>  | value to be written to the register |

**19.2.5.33** `static void FTM_HAL_SetChnOutputMask ( FTM_Type * ftmBase, uint8_t channel, bool mask ) [inline], [static]`

Parameters

|                |                                           |
|----------------|-------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer              |
| <i>channel</i> | The FTM peripheral channel number         |
| <i>mask</i>    | mask to be set 0 or 1, unmasked or masked |

**19.2.5.34** `static void FTM_HAL_SetChnOutputInitStateCmd ( FTM_Type * ftmBase, uint8_t channel, uint8_t state ) [inline], [static]`

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer      |
| <i>channel</i> | The FTM peripheral channel number |
| <i>state</i>   | counter value to be set 0 or 1    |

**19.2.5.35** `static void FTM_HAL_SetChnOutputPolarityCmd ( FTM_Type * ftmBase, uint8_t channel, uint8_t pol ) [inline], [static]`

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer      |
| <i>channel</i> | The FTM peripheral channel number |
| <i>pol</i>     | polarity to be set 0 or 1         |

**19.2.5.36** `static void FTM_HAL_SetChnFaultInputPolarityCmd ( FTM_Type * ftmBase,  
uint8_t channel, uint8_t pol ) [inline], [static]`

Parameters

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                     |
| <i>channel</i> | The FTM peripheral channel number                |
| <i>pol</i>     | polarity to be set, 0: active high, 1:active low |

**19.2.5.37** `static void FTM_HAL_EnableFaultInt ( FTM_Type * ftmBase ) [inline],  
[static]`

Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

**19.2.5.38** `static void FTM_HAL_DisableFaultInt ( FTM_Type * ftmBase ) [inline],  
[static]`

Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

**19.2.5.39** `static void FTM_HAL_SetFaultControlMode ( FTM_Type * ftmBase, uint8_t  
mode ) [inline], [static]`

Parameters

---

## FlexTimer HAL driver

|                   |                              |
|-------------------|------------------------------|
| <i>ftmBase</i>    | The FTM base address pointer |
| <i>mode,valid</i> | options are 1, 2, 3, 4       |

**19.2.5.40** `static void FTM_HAL_SetCaptureTestCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

Parameters

|                |                                                    |
|----------------|----------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                       |
| <i>enable</i>  | true to enable capture test mode, false to disable |

**19.2.5.41** `static void FTM_HAL_SetWriteProtectionCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

Parameters

|                |                                                                        |
|----------------|------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                           |
| <i>enable</i>  | true: Write-protection is enabled, false: Write-protection is disabled |

**19.2.5.42** `static void FTM_HAL_Enable ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

Parameters

|                |                                                                                                                           |
|----------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                              |
| <i>enable</i>  | true: all registers including FTM-specific registers are available false: only the TPM-compatible registers are available |

**19.2.5.43** `static void FTM_HAL_SetInitChnOutputCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

Parameters

|                |                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                        |
| <i>enable</i>  | true: the channels output is initialized according to the state of OUTINIT reg false: has no effect |

```
19.2.5.44 static void FTM_HAL_SetPwmSyncMode (FTM_Type * ftmBase, bool enable)
 [inline], [static]
```

## FlexTimer HAL driver

### Parameters

|                |                                                                                                                                                                                                  |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                                                                                                     |
| <i>enable</i>  | true: no restriction both software and hardware triggers can be used<br>false: software trigger can only be used for MOD and CnV synch, hardware trigger only for OUTMASK and FTM counter synch. |

**19.2.5.45** `static void FTM_HAL_SetSoftwareTriggerCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

### Parameters

|                |                                                                             |
|----------------|-----------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer.                                               |
| <i>enable</i>  | true: software trigger is selected, false: software trigger is not selected |

**19.2.5.46** `void FTM_HAL_SetHardwareSyncTriggerSrc ( FTM_Type * ftmBase, uint32_t trigger_num, bool enable )`

### Parameters

|                    |                                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i>     | The FTM base address pointer                                                                                                                             |
| <i>trigger_num</i> | 0, 1, 2 for trigger0, trigger1 and trigger3                                                                                                              |
| <i>enable</i>      | true: enable hardware trigger from field <i>trigger_num</i> for PWM synch<br>false: disable hardware trigger from field <i>trigger_num</i> for PWM synch |

**19.2.5.47** `static void FTM_HAL_SetOutmaskPwmSyncModeCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

### Parameters

|                |                                                                                                                                      |
|----------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                                         |
| <i>enable</i>  | true if OUTMASK register is updated only by PWM sync<br>false if OUTMASK register is updated in all rising edges of the system clock |

**19.2.5.48** `static void FTM_HAL_SetCountReinitSyncCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

Parameters

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                        |
| <i>enable</i>  | True to update FTM counter when triggered , false to count normally |

**19.2.5.49** `static void FTM_HAL_SetMaxLoadingCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

Parameters

|                |                                                        |
|----------------|--------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                           |
| <i>enable</i>  | True to enable maximum loading point, false to disable |

**19.2.5.50** `static void FTM_HAL_SetMinLoadingCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

Parameters

|                |                                                        |
|----------------|--------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                           |
| <i>enable</i>  | True to enable minimum loading point, false to disable |

**19.2.5.51** `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

**19.2.5.52** `static void FTM_HAL_SetDualChnFaultCmd ( FTM_Type * ftmBase, uint8_t chnlPairNum, bool enable ) [inline], [static]`

## FlexTimer HAL driver

### Parameters

|                    |                                                |
|--------------------|------------------------------------------------|
| <i>ftmBase</i>     | The FTM base address pointer                   |
| <i>chnlPairNum</i> | The FTM peripheral channel pair number         |
| <i>enable</i>      | True to enable fault control, false to disable |

**19.2.5.53** `static void FTM_HAL_SetDualChnPwmSyncCmd ( FTM_Type * ftmBase,  
uint8_t chnlPairNum, bool enable ) [inline], [static]`

### Parameters

|                    |                                                      |
|--------------------|------------------------------------------------------|
| <i>ftmBase</i>     | The FTM base address pointer                         |
| <i>chnlPairNum</i> | The FTM peripheral channel pair number               |
| <i>enable</i>      | True to enable PWM synchronization, false to disable |

**19.2.5.54** `static void FTM_HAL_SetDualChnDeadtimeCmd ( FTM_Type * ftmBase, uint8_t  
chnlPairNum, bool enable ) [inline], [static]`

### Parameters

|                    |                                                     |
|--------------------|-----------------------------------------------------|
| <i>ftmBase</i>     | The FTM base address pointer                        |
| <i>chnlPairNum</i> | The FTM peripheral channel pair number              |
| <i>enable</i>      | True to enable deadtime insertion, false to disable |

**19.2.5.55** `static void FTM_HAL_SetDualChnDecapCmd ( FTM_Type * ftmBase, uint8_t  
chnlPairNum, bool enable ) [inline], [static]`

### Parameters

|                    |                                                         |
|--------------------|---------------------------------------------------------|
| <i>ftmBase</i>     | The FTM base address pointer                            |
| <i>chnlPairNum</i> | The FTM peripheral channel pair number                  |
| <i>enable</i>      | True to enable dual edge capture mode, false to disable |

**19.2.5.56** `static void FTM_HAL_SetDualEdgeCaptureCmd ( FTM_Type * ftmBase, uint8_t  
chnlPairNum, bool enable ) [inline], [static]`



## Parameters

|                    |                                                    |
|--------------------|----------------------------------------------------|
| <i>ftmBase</i>     | The FTM base address pointer                       |
| <i>chnlPairNum</i> | The FTM peripheral channel pair number             |
| <i>enable</i>      | True to enable dual edge capture, false to disable |

**19.2.5.57** `static void FTM_HAL_SetDualChnCompCmd ( FTM_Type * ftmBase, uint8_t chnlPairNum, bool enable ) [inline], [static]`

## Parameters

|                    |                                                     |
|--------------------|-----------------------------------------------------|
| <i>ftmBase</i>     | The FTM base address pointer                        |
| <i>chnlPairNum</i> | The FTM peripheral channel pair number              |
| <i>enable</i>      | True to enable complementary mode, false to disable |

**19.2.5.58** `static void FTM_HAL_SetDualChnCombineCmd ( FTM_Type * ftmBase, uint8_t chnlPairNum, bool enable ) [inline], [static]`

## Parameters

|                    |                                                          |
|--------------------|----------------------------------------------------------|
| <i>ftmBase</i>     | The FTM base address pointer                             |
| <i>chnlPairNum</i> | The FTM peripheral channel pair number                   |
| <i>enable</i>      | True to enable channel pair to combine, false to disable |

**19.2.5.59** `static void FTM_HAL_SetDeadtimePrescale ( FTM_Type * ftmBase, ftm_deadtime_ps_t divider ) [inline], [static]`

## Parameters

|                |                                                                                             |
|----------------|---------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                |
| <i>divider</i> | The FTM peripheral prescale divider<br>0x :divided by 1, 10: divided by 4, 11:divided by 16 |

**19.2.5.60** `static void FTM_HAL_SetDeadtimeCount ( FTM_Type * ftmBase, uint8_t count ) [inline], [static]`

## FlexTimer HAL driver

### Parameters

|                |                                                                                                                  |
|----------------|------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                     |
| <i>count</i>   | The FTM peripheral prescale divider<br>0: no counts inserted, 1: 1 count is inserted, 2: 2 count is inserted.... |

### 19.2.5.61 static void FTM\_HAL\_SetInitTriggerCmd ( FTM\_Type \* *ftmBase*, bool *enable* ) [inline], [static]

### Parameters

|                |                                  |
|----------------|----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer     |
| <i>enable</i>  | True to enable, false to disable |

### 19.2.5.62 void FTM\_HAL\_SetChnTriggerCmd ( FTM\_Type \* *ftmBase*, uint8\_t *channel*, bool *val* )

Enables or disables 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>ftmBase</i> | The FTM base address pointer                        |
| <i>channel</i> | Channel to be enabled, valid value 0, 1, 2, 3, 4, 5 |
| <i>val</i>     | True to enable, false to disable                    |

### 19.2.5.63 static bool FTM\_HAL\_IsChnTriggerGenerated ( FTM\_Type \* *ftmBase* ) [inline], [static]

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

### Returns

true if there is a channel trigger event, false if not.

```
19.2.5.64 static uint8_t FTM_HAL_GetDetectedFaultInput (FTM_Type * ftmBase)
 [inline], [static]
```

This function reads the status for all fault inputs

## FlexTimer HAL driver

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

### Returns

Return fault byte

**19.2.5.65** `static bool FTM_HAL_IsWriteProtectionEnabled ( FTM_Type * ftmBase )`  
`[inline], [static]`

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

### Returns

True if enabled, false if not

**19.2.5.66** `static void FTM_HAL_SetQuadDecoderCmd ( FTM_Type * ftmBase, bool`  
`enable ) [inline], [static]`

### Parameters

|                |                                  |
|----------------|----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer     |
| <i>enable</i>  | True to enable, false to disable |

**19.2.5.67** `static void FTM_HAL_SetQuadPhaseAFilterCmd ( FTM_Type * ftmBase, bool`  
`enable ) [inline], [static]`

### Parameters

|                |                                                                |
|----------------|----------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                   |
| <i>enable</i>  | true enables the phase input filter, false disables the filter |

**19.2.5.68** `static void FTM_HAL_SetQuadPhaseBFilterCmd ( FTM_Type * ftmBase, bool`  
`enable ) [inline], [static]`

Parameters

|                |                                                                |
|----------------|----------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                   |
| <i>enable</i>  | true enables the phase input filter, false disables the filter |

**19.2.5.69 static void FTM\_HAL\_SetQuadPhaseAPolarity ( FTM\_Type \* *ftmBase*, *ftm\_quad\_phase\_polarity\_t mode* ) [inline], [static]**

Parameters

|                |                                          |
|----------------|------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer             |
| <i>mode</i>    | 0: Normal polarity, 1: Inverted polarity |

**19.2.5.70 static void FTM\_HAL\_SetQuadPhaseBPolarity ( FTM\_Type \* *ftmBase*, *ftm\_quad\_phase\_polarity\_t mode* ) [inline], [static]**

Parameters

|                |                                          |
|----------------|------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer             |
| <i>mode</i>    | 0: Normal polarity, 1: Inverted polarity |

**19.2.5.71 static void FTM\_HAL\_SetQuadMode ( FTM\_Type \* *ftmBase*, *ftm\_quad\_decode\_mode\_t quadMode* ) [inline], [static]**

Parameters

|                 |                                                                              |
|-----------------|------------------------------------------------------------------------------|
| <i>ftmBase</i>  | The FTM base address pointer                                                 |
| <i>quadMode</i> | 0: Phase A and Phase B encoding mode<br>1: Count and direction encoding mode |

**19.2.5.72 static uint8\_t FTM\_HAL\_GetQuadDir ( FTM\_Type \* *ftmBase* ) [inline], [static]**

## FlexTimer HAL driver

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

### Returns

1 if counting direction is increasing, 0 if counting direction is decreasing

**19.2.5.73** `static uint8_t FTM_HAL_GetQuadTimerOverflowDir ( FTM_Type * ftmBase )`  
`[inline], [static]`

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

### Returns

1 if TOF bit was set on the top of counting, 0 if TOF bit was set on the bottom of counting

**19.2.5.74** `void FTM_HAL_SetChnInputCaptureFilter ( FTM_Type * ftmBase, uint8_t`  
`channel, uint8_t val )`

### Parameters

|                |                                                                                |
|----------------|--------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                   |
| <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                                                         |

**19.2.5.75** `static void FTM_HAL_SetFaultInputFilterVal ( FTM_Type * ftmBase, uint32_t val`  
`) [inline], [static]`

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

|            |                          |
|------------|--------------------------|
| <i>val</i> | fault input filter value |
|------------|--------------------------|

**19.2.5.76** `static void FTM_HAL_SetFaultInputFilterCmd ( FTM_Type * ftmBase, uint8_t inputNum, bool val ) [inline], [static]`

Parameters

|                 |                                                                        |
|-----------------|------------------------------------------------------------------------|
| <i>ftmBase</i>  | The FTM base address pointer                                           |
| <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 |

**19.2.5.77** `static void FTM_HAL_SetFaultInputCmd ( FTM_Type * ftmBase, uint8_t inputNum, bool val ) [inline], [static]`

Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>ftmBase</i>  | The FTM base address pointer                             |
| <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 |

**19.2.5.78** `static void FTM_HAL_SetDualChnInvertCmd ( FTM_Type * ftmBase, uint8_t chnlPairNum, bool val ) [inline], [static]`

Parameters

|                    |                                                                  |
|--------------------|------------------------------------------------------------------|
| <i>ftmBase</i>     | The FTM base address pointer                                     |
| <i>chnlPairNum</i> | The FTM peripheral channel pair number                           |
| <i>val</i>         | true to enable channel inverting, false to disable channel inver |

**19.2.5.79** `static void FTM_HAL_SetInvctrlReg ( FTM_Type * ftmBase, uint32_t regVal ) [inline], [static]`

This function is enable/disable inverting control on multiple channel pairs.

## FlexTimer HAL driver

### Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer        |
| <i>regVal</i>  | value to be written to the register |

**19.2.5.80** `static void FTM_HAL_SetChnSoftwareCtrlCmd ( FTM_Type * ftmBase, uint8_t channel, bool val ) [inline], [static]`

### Parameters

|                |                                                                                                                              |
|----------------|------------------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                                 |
| <i>channel</i> | Channel to be enabled or disabled                                                                                            |
| <i>val</i>     | true to enable, channel output will be affected by software output control<br>false to disable, channel output is unaffected |

**19.2.5.81** `static void FTM_HAL_SetChnSoftwareCtrlVal ( FTM_Type * ftmBase, uint8_t channel, bool val ) [inline], [static]`

### Parameters

|                |                               |
|----------------|-------------------------------|
| <i>ftmBase</i> | The FTM base address pointer. |
| <i>channel</i> | Channel to be configured      |
| <i>val</i>     | True to set 1, false to set 0 |

**19.2.5.82** `static void FTM_HAL_SetPwmLoadCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

### Parameters

|                |                                  |
|----------------|----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer     |
| <i>enable</i>  | true to enable, false to disable |

**19.2.5.83** `static void FTM_HAL_SetPwmLoadChnSelCmd ( FTM_Type * ftmBase, uint8_t channel, bool val ) [inline], [static]`



Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                         |
| <i>channel</i> | Channel to be configured                                                                                             |
| <i>val</i>     | true means include the channel in the matching process<br>false means do not include channel in the matching process |

**19.2.5.84** `static void FTM_HAL_SetGlobalTimeBaseOutputCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

Parameters

|                |                                  |
|----------------|----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer     |
| <i>enable</i>  | True to enable, false to disable |

**19.2.5.85** `static void FTM_HAL_SetGlobalTimeBaseCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

Parameters

|                |                                  |
|----------------|----------------------------------|
| <i>ftmBase</i> | The FTM base address pointer     |
| <i>enable</i>  | True to enable, false to disable |

**19.2.5.86** `static void FTM_HAL_SetBdmMode ( FTM_Type * ftmBase, ftm_bdm_mode_t val ) [inline], [static]`

Parameters

|                |                                                                                        |
|----------------|----------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                           |
| <i>val</i>     | FTM behaviour in BDM mode, options are defined in the enum <code>ftm_bdm_mode_t</code> |

**19.2.5.87** `static void FTM_HAL_SetTofFreq ( FTM_Type * ftmBase, uint8_t val ) [inline], [static]`

## FlexTimer HAL driver

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer       |
| <i>val</i>     | Value of the TOF bit set frequency |

### 19.2.5.88 void FTM\_HAL\_SetSyncMode ( FTM\_Type \* *ftmBase*, uint32\_t *syncMethod* )

This function will set the necessary bits for the synchronization mode that user wishes to use.

Parameters

|                   |                                                                                                                           |
|-------------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i>    | The FTM base address pointer                                                                                              |
| <i>syncMethod</i> | Synchronization method defined by <i>ftm_sync_method_t</i> enum. User can choose multiple synch methods by OR'ing options |

### 19.2.5.89 static void FTM\_HAL\_SetSwoctrlHardwareSyncModeCmd ( FTM\_Type \* *ftmBase*, bool *enable* ) [inline], [static]

Parameters

|                |                                                                                                                              |
|----------------|------------------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                                 |
| <i>enable</i>  | true means the hardware trigger activates register sync<br>false means the hardware trigger does not activate register sync. |

### 19.2.5.90 static void FTM\_HAL\_SetInvctrlHardwareSyncModeCmd ( FTM\_Type \* *ftmBase*, bool *enable* ) [inline], [static]

Parameters

|                |                                                                                                                              |
|----------------|------------------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                                 |
| <i>enable</i>  | true means the hardware trigger activates register sync<br>false means the hardware trigger does not activate register sync. |

### 19.2.5.91 static void FTM\_HAL\_SetOutmaskHardwareSyncModeCmd ( FTM\_Type \* *ftmBase*, bool *enable* ) [inline], [static]

## Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                         |
| <i>enable</i>  | true means hardware trigger activates register sync<br>false means hardware trigger does not activate register sync. |

**19.2.5.92 static void FTM\_HAL\_SetModCntinCvHardwareSyncModeCmd ( FTM\_Type \* *ftmBase*, bool *enable* ) [inline], [static]**

## Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                         |
| <i>enable</i>  | true means hardware trigger activates register sync<br>false means hardware trigger does not activate register sync. |

**19.2.5.93 static void FTM\_HAL\_SetCounterHardwareSyncModeCmd ( FTM\_Type \* *ftmBase*, bool *enable* ) [inline], [static]**

## Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                         |
| <i>enable</i>  | true means hardware trigger activates register sync<br>false means hardware trigger does not activate register sync. |

**19.2.5.94 static void FTM\_HAL\_SetSwoctrlSoftwareSyncModeCmd ( FTM\_Type \* *ftmBase*, bool *enable* ) [inline], [static]**

## Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                         |
| <i>enable</i>  | true means software trigger activates register sync<br>false means software trigger does not activate register sync. |

**19.2.5.95 static void FTM\_HAL\_SetInvctrlSoftwareSyncModeCmd ( FTM\_Type \* *ftmBase*, bool *enable* ) [inline], [static]**

## FlexTimer HAL driver

### Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                         |
| <i>enable</i>  | true means software trigger activates register sync<br>false means software trigger does not activate register sync. |

**19.2.5.96** `static void FTM_HAL_SetOutmaskSoftwareSyncModeCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

### Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                         |
| <i>enable</i>  | true means software trigger activates register sync<br>false means software trigger does not activate register sync. |

**19.2.5.97** `static void FTM_HAL_SetModCntrCvSoftwareSyncModeCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

### Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                         |
| <i>enable</i>  | true means software trigger activates register sync<br>false means software trigger does not activate register sync. |

**19.2.5.98** `static void FTM_HAL_SetCounterSoftwareSyncModeCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

### Parameters

|                |                                                                                                                      |
|----------------|----------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                         |
| <i>enable</i>  | true means software trigger activates register sync<br>false means software trigger does not activate register sync. |

**19.2.5.99** `static void FTM_HAL_SetPwmSyncModeCmd ( FTM_Type * ftmBase, bool enable ) [inline], [static]`

## Parameters

|                |                                                                               |
|----------------|-------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                  |
| <i>enable</i>  | true means use Enhanced PWM synchronization<br>false means to use Legacy mode |

**19.2.5.100** `static void FTM_HAL_SetSwoctrlPwmSyncModeCmd ( FTM_Type * ftmBase,  
bool enable ) [inline], [static]`

## Parameters

|                |                                                                                                                                    |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                                       |
| <i>enable</i>  | true means SWOCTRL register is updated by PWM synch<br>false means SWOCTRL register is updated at all rising edges of system clock |

**19.2.5.101** `static void FTM_HAL_SetInvctrlPwmSyncModeCmd ( FTM_Type * ftmBase,  
bool enable ) [inline], [static]`

## Parameters

|                |                                                                                                                                    |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                                       |
| <i>enable</i>  | true means INVCTRL register is updated by PWM synch<br>false means INVCTRL register is updated at all rising edges of system clock |

**19.2.5.102** `static void FTM_HAL_SetCntinPwmSyncModeCmd ( FTM_Type * ftmBase,  
bool enable ) [inline], [static]`

## Parameters

|                |                                                                                                                                |
|----------------|--------------------------------------------------------------------------------------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                                                                                                   |
| <i>enable</i>  | true means CNTIN register is updated by PWM synch<br>false means CNTIN register is updated at all rising edges of system clock |

**19.2.5.103** `void FTM_HAL_Reset ( FTM_Type * ftmBase )`

## FlexTimer HAL driver

Parameters

|                |                              |
|----------------|------------------------------|
| <i>ftmBase</i> | The FTM base address pointer |
|----------------|------------------------------|

### 19.2.5.104 void FTM\_HAL\_Init ( FTM\_Type \* *ftmBase* )

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>ftmBase</i> | The FTM base address pointer. |
|----------------|-------------------------------|

### 19.2.5.105 void FTM\_HAL\_EnablePwmMode ( FTM\_Type \* *ftmBase*, ftm\_pwm\_param\_t \* *config*, uint8\_t *channel* )

Parameters

|                |                                                    |
|----------------|----------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                       |
| <i>config</i>  | PWM configuration parameter                        |
| <i>channel</i> | The channel or channel pair number(combined mode). |

### 19.2.5.106 void FTM\_HAL\_DisablePwmMode ( FTM\_Type \* *ftmBase*, ftm\_pwm\_param\_t \* *config*, uint8\_t *channel* )

Parameters

|                |                                                    |
|----------------|----------------------------------------------------|
| <i>ftmBase</i> | The FTM base address pointer                       |
| <i>config</i>  | PWM configuration parameter                        |
| <i>channel</i> | The channel or channel pair number(combined mode). |

## 19.3 FlexTimer Peripheral Driver

### 19.3.1 Overview

This section describes the programming interface of the FlexTimer Peripheral driver.

### 19.3.2 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.

### 19.3.3 FlexTimer Initialization

1. To initialize the FlexTimer driver, call the `FTM_DRV_Init()` function and pass the instance number of the relevant FTM. 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 tofFrequency;
 bool isFTMMode;
 uint8_t BDMMode;
 bool isWriteProtection;

 bool isTimerOverFlowInterrupt;
 bool isFaultInterrupt;
} ftm_user_config_t;
```

### 19.3.4 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;

 uint16_t uFirstEdgeDelayPercent;
} ftm_pwm_param_t;
```

0 = inactive signal (0% duty cycle)...  
100 = active signal (100% duty cycle). //

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//

## FlexTimer Peripheral Driver

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 is delayed.

### Data Structures

- struct `ftm_user_config_t`  
*Configuration structure that the user needs to set. [More...](#)*

### Functions

- `ftm_status_t FTM_DRV_Init` (uint32\_t instance, const `ftm_user_config_t` \*info)  
*Initializes the FTM driver.*
- void `FTM_DRV_Deinit` (uint32\_t instance)  
*Shuts down the FTM driver.*
- void `FTM_DRV_PwmStop` (uint32\_t instance, `ftm_pwm_param_t` \*param, uint8\_t channel)  
*Stops the channel PWM.*
- `ftm_status_t FTM_DRV_PwmStart` (uint32\_t instance, `ftm_pwm_param_t` \*param, uint8\_t channel)  
*Configures the duty cycle and frequency and starts outputting the PWM on a specified channel.*
- void `FTM_DRV_QuadDecodeStart` (uint32\_t instance, `ftm_phase_params_t` \*phaseAParams, `ftm_phase_params_t` \*phaseBParams, `ftm_quad_decode_mode_t` quadMode)  
*Configures the parameters and activates the quadrature decode mode.*
- void `FTM_DRV_QuadDecodeStop` (uint32\_t instance)  
*De-activates the quadrature decode mode.*
- void `FTM_DRV_CounterStart` (uint32\_t instance, `ftm_counting_mode_t` countMode, uint32\_t countStartVal, uint32\_t countFinalVal, bool enableOverflowInt)  
*Starts the FTM counter.*
- void `FTM_DRV_CounterStop` (uint32\_t instance)  
*Stops the FTM counter.*
- uint32\_t `FTM_DRV_CounterRead` (uint32\_t instance)  
*Reads back the current value of the FTM counter.*
- void `FTM_DRV_SetClock` (uint8\_t instance, `ftm_clock_source_t` clock, `ftm_clock_ps_t` clockPs)  
*Set FTM clock source.*
- uint32\_t `FTM_DRV_GetClock` (uint8\_t instance)  
*Retrieves the frequency of the clock source feeding 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_SetupChnInputCapture` (uint32\_t instance, `ftm_input_capture_edge_mode_t` captureMode, uint8\_t channel, uint8\_t filterVal)  
*Enables capture of an input signal on the channel using the function parameters.*
- void `FTM_DRV_SetupChnOutputCompare` (uint32\_t instance, `ftm_output_compare_edge_mode_t` compareMode, uint8\_t channel, uint32\_t compareVal)



- Configures the FTM to generate timed pulses.*

• void [FTM\\_DRV\\_SetupChnDualEdgeCapture](#) (uint32\_t instance, [ftm\\_dual\\_edge\\_capture\\_param\\_t](#) \*param, uint8\_t channel, uint8\_t filterVal)

*Configures the dual edge capture mode of the FTM.*
- void [FTM\\_DRV\\_IRQHandler](#) (uint32\_t instance)

*Action to take when an FTM interrupt is triggered.*

### Variables

- FTM\_Type \*const [g\\_ftmBase](#) [FTM\_INSTANCE\_COUNT]

*Table of base addresses for FTM instances.*
- const IRQn\_Type [g\\_ftmIrqId](#) [FTM\_INSTANCE\_COUNT]

*Table to save FTM IRQ enumeration numbers defined in the CMSIS header file.*

## 19.3.5 Data Structure Documentation

### 19.3.5.1 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.*

## 19.3.6 Function Documentation

### 19.3.6.1 [ftm\\_status\\_t](#) [FTM\\_DRV\\_Init](#) ( uint32\_t *instance*, const [ftm\\_user\\_config\\_t](#) \* *info* )

#### Parameters

|                 |                                                                               |
|-----------------|-------------------------------------------------------------------------------|
| <i>instance</i> | The FTM peripheral instance number.                                           |
| <i>info</i>     | The FTM user configuration structure, see <a href="#">ftm_user_config_t</a> . |

#### Returns

kStatusFtmSuccess means success, otherwise means failed.

### 19.3.6.2 void [FTM\\_DRV\\_Deinit](#) ( uint32\_t *instance* )

## FlexTimer Peripheral Driver

### Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The FTM peripheral instance number. |
|-----------------|-------------------------------------|

### 19.3.6.3 void FTM\_DRV\_PwmStop ( uint32\_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 finds the channel pair associated with the channel number passed in. |

### 19.3.6.4 ftm\_status\_t FTM\_DRV\_PwmStart ( uint32\_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 finds the channel pair associated with the channel number passed in. |

### Returns

kStatusFtmSuccess if the PWM setup was successful, kStatusFtmError on failure as the PWM counter is disabled

### 19.3.6.5 void FTM\_DRV\_QuadDecodeStart ( uint32\_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 |

### 19.3.6.6 void FTM\_DRV\_QuadDecodeStop ( uint32\_t instance )

Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>instance</i> | Instance number of the FTM module. |
|-----------------|------------------------------------|

### 19.3.6.7 void FTM\_DRV\_CounterStart ( uint32\_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 the up-counting option and provide 0x0 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 ftm_counting_mode_t.     |
| <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    |

### 19.3.6.8 void FTM\_DRV\_CounterStop ( uint32\_t instance )

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The FTM peripheral instance number. |
|-----------------|-------------------------------------|

### 19.3.6.9 uint32\_t FTM\_DRV\_CounterRead ( uint32\_t instance )

## FlexTimer Peripheral Driver

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The FTM peripheral instance number. |
|-----------------|-------------------------------------|

### 19.3.6.10 void FTM\_DRV\_SetClock ( uint8\_t *instance*, ftm\_clock\_source\_t *clock*, ftm\_clock\_ps\_t *clockPs* )

This function will save the users clock source selection in the driver and uses this to set the clock source whenever the user decides to use features provided by this driver like counter, PWM generation etc. It will also set the clock divider.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The FTM peripheral instance number.        |
| <i>clock</i>    | The clock source to use, cannot pick None. |
| <i>clockPs</i>  | The clock divider value.                   |

### 19.3.6.11 uint32\_t FTM\_DRV\_GetClock ( uint8\_t *instance* )

Function will return a 0 if no clock source is selected and the FTM counter is disabled

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The FTM peripheral instance number. |
|-----------------|-------------------------------------|

Returns

The frequency of the clock source running the FTM counter, returns 0 if counter is disabled

### 19.3.6.12 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 |
|-----------------------|-----------------------------------------------------------|

**19.3.6.13 void FTM\_DRV\_SetFaultIntCmd ( uint32\_t instance, bool faultEnable )**

Parameters

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <i>instance</i>    | The FTM peripheral instance number.              |
| <i>faultEnable</i> | true: enable the fault interrupt, false: disable |

**19.3.6.14 void FTM\_DRV\_SetupChnInputCapture ( uint32\_t instance, ftm\_input\_capture\_edge\_mode\_t captureMode, uint8\_t channel, uint8\_t filterVal )**

When the edge specified in the captureMode argument occurs on the channel the FTM counter is captured into the CnV register. The user has to read the CnV register separately to get this value. The filter function is disabled if the filterVal argument passed in is 0. The filter function is available only for 0, 1, 2, 3 channels.

Parameters

|                    |                                                                                       |
|--------------------|---------------------------------------------------------------------------------------|
| <i>instance</i>    | The FTM peripheral instance number                                                    |
| <i>captureMode</i> | Specifies which edge to capture                                                       |
| <i>channel</i>     | The channel number                                                                    |
| <i>filterVal</i>   | Filter value to be used, specify 0 to disable filter. Available only for channels 0-3 |

**19.3.6.15 void FTM\_DRV\_SetupChnOutputCompare ( uint32\_t instance, ftm\_output\_compare\_edge\_mode\_t compareMode, uint8\_t channel, uint32\_t compareVal )**

When the FTM counter matches the value of compareVal argument (this is written into CnV reg), the channel output is changed based on what is specified in the compareMode argument.

Parameters

|                    |                                                                        |
|--------------------|------------------------------------------------------------------------|
| <i>instance</i>    | The FTM peripheral instance number.                                    |
| <i>compareMode</i> | Action to take on the channel output when the compare condition is met |
| <i>channel</i>     | The channel number                                                     |
| <i>compareVal</i>  | Value to be programmed in the CnV register.                            |

## FlexTimer Peripheral Driver

**19.3.6.16 void FTM\_DRV\_SetupChnDualEdgeCapture ( uint32\_t *instance*,  
ftm\_dual\_edge\_capture\_param\_t \* *param*, uint8\_t *channel*, uint8\_t *filterVal* )**

This function sets up the dual edge capture mode on a channel pair. The capture edge for the channel pair and the capture mode (one-shot or continuous) is specified in the parameter argument. The filter function is disabled if the filterVal argument passed in is 0. The filter function is available only on channels 0 and 2. The user has to read the channel CnV registers separately to get the capture values.

Parameters

|                  |                                                                                                   |
|------------------|---------------------------------------------------------------------------------------------------|
| <i>instance</i>  | The FTM peripheral instance number.                                                               |
| <i>param</i>     | Controls the dual edge capture function                                                           |
| <i>channel</i>   | The channel number, the code finds the channel pair associated with the channel number passed in. |
| <i>filterVal</i> | Filter value to be used, specify 0 to disable filter. Available only for channels 0, 2            |

**19.3.6.17 void FTM\_DRV\_IRQHandler ( uint32\_t *instance* )**

The timer overflow flag is checked and cleared if set.

Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>instance</i> | Instance number of the FTM module. |
|-----------------|------------------------------------|

## 19.3.7 Variable Documentation

**19.3.7.1 FTM\_Type\* const g\_ftmBase[FTM\_INSTANCE\_COUNT]**

**19.3.7.2 const IRQn\_Type g\_ftmIrqld[FTM\_INSTANCE\_COUNT]**

---

## Chapter 20

# General Purpose Input/Output (GPIO)

### 20.1 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

### 20.2 GPIO HAL driver

#### 20.2.1 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](#) = 0U,  
  [kGpioDigitalOutput](#) = 1U }  
*GPIO direction definition.*

#### Configuration

- void [GPIO\\_HAL\\_SetPinDir](#) (GPIO\_Type \*base, uint32\_t pin, [gpio\\_pin\\_direction\\_t](#) direction)  
*Sets the individual GPIO pin to general input or output.*
- static void [GPIO\\_HAL\\_SetPortDir](#) (GPIO\_Type \*base, uint32\_t pinDirectionMap)  
*Sets the GPIO port pins to general input or output.*

#### Status

- static [gpio\\_pin\\_direction\\_t](#) [GPIO\\_HAL\\_GetPinDir](#) (GPIO\_Type \*base, uint32\_t pin)  
*Gets the current direction of the individual GPIO pin.*
- static uint32\_t [GPIO\\_HAL\\_GetPortDir](#) (GPIO\_Type \*base)  
*Gets the GPIO port pins direction.*

#### Output Operation

- void [GPIO\\_HAL\\_WritePinOutput](#) (GPIO\_Type \*base, 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](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current pin output.*
- static void [GPIO\\_HAL\\_SetPinOutput](#) (GPIO\_Type \*base, uint32\_t pin)  
*Sets the output level of the individual GPIO pin to logic 1.*
- static void [GPIO\\_HAL\\_ClearPinOutput](#) (GPIO\_Type \*base, uint32\_t pin)  
*Clears the output level of the individual GPIO pin to logic 0.*
- static void [GPIO\\_HAL\\_TogglePinOutput](#) (GPIO\_Type \*base, uint32\_t pin)  
*Reverses the current output logic of the individual GPIO pin.*



- static void **GPIO\_HAL\_WritePortOutput** (GPIO\_Type \*base, uint32\_t portOutput)  
*Sets the output of the GPIO port pins to a specific logic value.*
- static uint32\_t **GPIO\_HAL\_ReadPortOutput** (GPIO\_Type \*base)  
*Reads out all pin output status of the current port.*
- static void **GPIO\_HAL\_SetPortOutput** (GPIO\_Type \*base, uint32\_t portOutput)  
*Sets the output level of the GPIO port pins to logic 1.*
- static void **GPIO\_HAL\_ClearPortOutput** (GPIO\_Type \*base, uint32\_t portOutput)  
*Clears the output level of the GPIO port pins to logic 0.*
- static void **GPIO\_HAL\_TogglePortOutput** (GPIO\_Type \*base, uint32\_t portOutput)  
*Reverses the current output logic of the GPIO port pins.*

## Input Operation

- static uint32\_t **GPIO\_HAL\_ReadPinInput** (GPIO\_Type \*base, uint32\_t pin)  
*Reads the current input value of the individual GPIO pin.*
- static uint32\_t **GPIO\_HAL\_ReadPortInput** (GPIO\_Type \*base)  
*Reads the current input value of a specific GPIO port.*

## 20.2.2 Enumeration Type Documentation

### 20.2.2.1 enum gpio\_pin\_direction\_t

Enumerator

- kGpioDigitalInput*** Set current pin as digital input.
- kGpioDigitalOutput*** Set current pin as digital output.

## 20.2.3 Function Documentation

### 20.2.3.1 void GPIO\_HAL\_SetPinDir ( GPIO\_Type \* base, uint32\_t pin, gpio\_pin\_direction\_t direction )

Parameters

|                  |                                                                                                                                                  |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | GPIO base pointer(PTA, PTB, PTC, etc.)                                                                                                           |
| <i>pin</i>       | GPIO port pin number                                                                                                                             |
| <i>direction</i> | GPIO directions <ul style="list-style-type: none"> <li>• kGpioDigitalInput: set to input</li> <li>• kGpioDigitalOutput: set to output</li> </ul> |

## GPIO HAL driver

**20.2.3.2** `static void GPIO_HAL_SetPortDir ( GPIO_Type * base, uint32_t pinDirectionMap ) [inline], [static]`

This function operates all 32 port pins.

## Parameters

|                         |                                                                                                                                                                      |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>             | GPIO base pointer (PTA, PTB, PTC, etc.)                                                                                                                              |
| <i>pinDirection-Map</i> | GPIO directions bit map <ul style="list-style-type: none"> <li>• 0: set to input</li> <li>• 1: set to output</li> <li>• LSB: pin 0</li> <li>• MSB: pin 31</li> </ul> |

**20.2.3.3 static gpio\_pin\_direction\_t GPIO\_HAL\_GetPinDir ( GPIO\_Type \* *base*, uint32\_t *pin* ) [inline], [static]**

## Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | GPIO base pointer(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.

**20.2.3.4 static uint32\_t GPIO\_HAL\_GetPortDir ( GPIO\_Type \* *base* ) [inline], [static]**

This function gets all 32-pin directions as a 32-bit integer.

## Parameters

|             |                                         |
|-------------|-----------------------------------------|
| <i>base</i> | GPIO base pointer (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

## GPIO HAL driver

20.2.3.5 void GPIO\_HAL\_WritePinOutput ( GPIO\_Type \* *base*, uint32\_t *pin*, uint32\_t *output* )

## Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | GPIO base pointer(PTA, PTB, PTC, etc.) |
| <i>pin</i>    | GPIO port pin number                   |
| <i>output</i> | pin output logic level                 |

**20.2.3.6** `static uint32_t GPIO_HAL_ReadPinOutput ( GPIO_Type * base, uint32_t pin )`  
**[inline], [static]**

## Parameters

|             |                                         |
|-------------|-----------------------------------------|
| <i>base</i> | GPIO base pointer (PTA, PTB, PTC, etc.) |
| <i>pin</i>  | GPIO port pin number                    |

## Returns

current pin output status. 0 - Low logic, 1 - High logic

**20.2.3.7** `static void GPIO_HAL_SetPinOutput ( GPIO_Type * base, uint32_t pin )`  
**[inline], [static]**

## Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | GPIO base pointer(PTA, PTB, PTC, etc.) |
| <i>pin</i>  | GPIO port pin number                   |

**20.2.3.8** `static void GPIO_HAL_ClearPinOutput ( GPIO_Type * base, uint32_t pin )`  
**[inline], [static]**

## Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | GPIO base pointer(PTA, PTB, PTC, etc.) |
| <i>pin</i>  | GPIO port pin number                   |

**20.2.3.9** `static void GPIO_HAL_TogglePinOutput ( GPIO_Type * base, uint32_t pin )`  
**[inline], [static]**

## GPIO HAL driver

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | GPIO base pointer(PTA, PTB, PTC, etc.) |
| <i>pin</i>  | GPIO port pin number                   |

### 20.2.3.10 `static void GPIO_HAL_WritePortOutput ( GPIO_Type * base, uint32_t portOutput ) [inline], [static]`

This function operates all 32 port pins.

Parameters

|                   |                                                                                                                                                                                                                                                   |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | GPIO base pointer (PTA, PTB, PTC, etc.)                                                                                                                                                                                                           |
| <i>portOutput</i> | data to configure the GPIO output. Each bit represents one pin. For each bit: <ul style="list-style-type: none"> <li>• 0: set logic level 0 to pin</li> <li>• 1: set logic level 1 to pin</li> <li>• LSB: pin 0</li> <li>• MSB: pin 31</li> </ul> |

### 20.2.3.11 `static uint32_t GPIO_HAL_ReadPortOutput ( GPIO_Type * base ) [inline], [static]`

This function operates all 32 port pins.

Parameters

|             |                                         |
|-------------|-----------------------------------------|
| <i>base</i> | GPIO base pointer (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

### 20.2.3.12 `static void GPIO_HAL_SetPortOutput ( GPIO_Type * base, uint32_t portOutput ) [inline], [static]`

This function operates all 32 port pins.

Parameters

|                   |                                                                                                                                                                                                                                                                 |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | GPIO base pointer(PTA, PTB, PTC, etc.)                                                                                                                                                                                                                          |
| <i>portOutput</i> | GPIO output port pin mask. Each bit represents one pin. For each bit: <ul style="list-style-type: none"> <li>• 0: pin output will not be changed.</li> <li>• 1: pin output will be set to logic level 1</li> <li>• LSB: pin 0</li> <li>• MSB: pin 31</li> </ul> |

**20.2.3.13 static void GPIO\_HAL\_ClearPortOutput ( GPIO\_Type \* *base*, uint32\_t *portOutput* ) [inline], [static]**

This function operates all 32 port pins.

Parameters

|                   |                                                                                                                                                                                                                                                                |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | GPIO base pointer(PTA, PTB, PTC, etc.)                                                                                                                                                                                                                         |
| <i>portOutput</i> | mask of GPIO output pins. Each bit represents one pin. For each bit: <ul style="list-style-type: none"> <li>• 0: pin output will not be changed.</li> <li>• 1: pin output will be set to logic level 0</li> <li>• LSB: pin 0</li> <li>• MSB: pin 31</li> </ul> |

**20.2.3.14 static void GPIO\_HAL\_TogglePortOutput ( GPIO\_Type \* *base*, uint32\_t *portOutput* ) [inline], [static]**

This function operates all 32 port pins.

Parameters

|                   |                                                                                                                                                                                                                                                                 |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | GPIO base pointer(PTA, PTB, PTC, etc.)                                                                                                                                                                                                                          |
| <i>portOutput</i> | mask of GPIO output pins. Each bit represents one pin. For each bit: <ul style="list-style-type: none"> <li>• 0: pin output will not be changed.</li> <li>• 1: pin output logic level will be reversed.</li> <li>• LSB: pin 0</li> <li>• MSB: pin 31</li> </ul> |

---

**GPIO HAL driver**

**20.2.3.15** `static uint32_t GPIO_HAL_ReadPinInput ( GPIO_Type * base, uint32_t pin )`  
`[inline], [static]`



Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | GPIO base pointer(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

**20.2.3.16** `static uint32_t GPIO_HAL_ReadPortInput ( GPIO_Type * base ) [inline], [static]`

This function gets all 32-pin input as a 32-bit integer.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | GPIO base pointer(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

## GPIO Peripheral driver

### 20.3 GPIO Peripheral driver

#### 20.3.1 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.

#### 20.3.2 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[];
```

#### 20.3.3 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.

This is an 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,
 }
}
```

```

};

// 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);

```

### 20.3.4 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.

## GPIO Peripheral driver

### 20.3.5 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);
```

### 20.3.6 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

- `GPIO_Type *const g_gpioBase [GPIO_INSTANCE_COUNT]`  
*Table of base addresses for GPIO instances.*
- `PORT_Type *const g_portBase [PORT_INSTANCE_COUNT]`  
*Table of base addresses for PORT instances.*

## 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.*

## GPIO Peripheral driver

### Interrupt

- bool [GPIO\\_DRV\\_IsPinIntPending](#) (uint32\_t pinName)  
*Reads the individual pin-interrupt status flag.*
- void [GPIO\\_DRV\\_ClearPinIntFlag](#) (uint32\_t pinName)  
*Clears the individual GPIO pin interrupt status flag.*

## 20.3.7 Data Structure Documentation

### 20.3.7.1 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.

### 20.3.7.2 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.*

#### 20.3.7.2.0.33 Field Documentation

##### 20.3.7.2.0.33.1 uint32\_t gpio\_output\_pin\_t::outputLogic

### 20.3.7.3 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.*

### 20.3.7.3.0.34 Field Documentation

20.3.7.3.0.34.1 `uint32_t gpio_input_pin_user_config_t::pinName`

20.3.7.3.0.34.2 `gpio_input_pin_t gpio_input_pin_user_config_t::config`

### 20.3.7.4 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.*

### 20.3.7.4.0.35 Field Documentation

20.3.7.4.0.35.1 `uint32_t gpio_output_pin_user_config_t::pinName`

20.3.7.4.0.35.2 `gpio_output_pin_t gpio_output_pin_user_config_t::config`

### 20.3.8 Macro Definition Documentation

20.3.8.1 `#define GPIO_PINS_OUT_OF_RANGE (0xFFFFFFFFU)`

20.3.8.2 `#define GPIO_PORT_SHIFT (0x8U)`

20.3.8.3 `#define GPIO_MAKE_PIN( r, p ) (((r) << GPIO_PORT_SHIFT) | (p))`

20.3.8.4 `#define GPIO_EXTRACT_PORT( v ) (((v) >> GPIO_PORT_SHIFT) & 0xFFU)`

20.3.8.5 `#define GPIO_EXTRACT_PIN( v ) ((v) & 0xFFU)`

### 20.3.9 Function Documentation

20.3.9.1 `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:

## GPIO Peripheral driver

```
// 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. |

### 20.3.9.2 void GPIO\_DRV\_InputPinInit ( const gpio\_input\_pin\_user\_config\_t \* *inputPin* )

#### Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>inputPin</i> | input GPIO pins pointer. |
|-----------------|--------------------------|

### 20.3.9.3 void GPIO\_DRV\_OutputPinInit ( const gpio\_output\_pin\_user\_config\_t \* *outputPin* )

#### Parameters

|                  |                           |
|------------------|---------------------------|
| <i>outputPin</i> | output GPIO pins pointer. |
|------------------|---------------------------|

### 20.3.9.4 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.



20.3.9.5 void GPIO\_DRV\_SetPinDir ( uint32\_t *pinName*, gpio\_pin\_direction\_t *direction* )

## GPIO Peripheral driver

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"> <li>• kGpioDigitalInput: corresponding pin is set as digital input.</li> <li>• kGpioDigitalOutput: corresponding pin is set as digital output.</li> </ul> |

### 20.3.9.6 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"> <li>• 0: corresponding pin output low logic level.</li> <li>• Non-0: corresponding pin output high logic level.</li> </ul> |

### 20.3.9.7 void GPIO\_DRV\_SetPinOutput ( uint32\_t *pinName* )

Parameters

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| <i>pinName</i> | GPIO pin name defined by the user in the GPIO pin enumeration list. |
|----------------|---------------------------------------------------------------------|

### 20.3.9.8 void GPIO\_DRV\_ClearPinOutput ( uint32\_t *pinName* )

Parameters

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| <i>pinName</i> | GPIO pin name defined by the user in the GPIO pin enumeration list. |
|----------------|---------------------------------------------------------------------|

### 20.3.9.9 void GPIO\_DRV\_TogglePinOutput ( uint32\_t *pinName* )

Parameters

---

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| <i>pinName</i> | GPIO pin name defined by the user in the GPIO pin enumeration list. |
|----------------|---------------------------------------------------------------------|

**20.3.9.10 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.

**20.3.9.11 bool GPIO\_DRV\_IsPinIntPending ( uint32\_t *pinName* )**

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>pinName</i> | GPIO pin name defined by the user in the GPIO pin enumeration list. |
|----------------|---------------------------------------------------------------------|

Returns

current pin interrupt status flag

- 0: interrupt is not detected.
- 1: interrupt is detected.

**20.3.9.12 void GPIO\_DRV\_ClearPinIntFlag ( uint32\_t *pinName* )**

Parameters

|                |                                                                     |
|----------------|---------------------------------------------------------------------|
| <i>pinName</i> | GPIO pin name defined by the user in the GPIO pin enumeration list. |
|----------------|---------------------------------------------------------------------|

## GPIO Peripheral driver

### 20.3.10 Variable Documentation

20.3.10.1 `GPIO_Type* const g_gpioBase[GPIO_INSTANCE_COUNT]`

20.3.10.2 `PORT_Type* const g_portBase[PORT_INSTANCE_COUNT]`

## Chapter 21

# Inter-Integrated Circuit (I2C)

### 21.1 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](#)

## I2C HAL driver

### 21.2 I2C HAL driver

#### 21.2.1 Overview

This section describes the programming interface of the I2C HAL driver.

#### 21.2.2 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                  |

#### 21.2.3 I2C Clock rate formulas

I2C baud rate =  $\text{bus\_clock\_Hz} / (\text{mult} * \text{SCL\_divider})$

SDA hold time =  $\text{bus\_clock\_period\_s} * \text{mult} * \text{SDA\_hold\_value}$

SCL start hold time =  $\text{bus\_clock\_period\_s} * \text{mult} * \text{SCL\_start\_hold\_value}$

SCL stop hold time =  $\text{bus\_clock\_period\_s} * \text{mult} * \text{SCL\_stop\_hold\_value}$

## 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` (I2C\_Type \*base)  
*Restores the I2C peripheral to reset state.*
- static void `I2C_HAL_Enable` (I2C\_Type \*base)  
*Enables the I2C module operation.*
- static void `I2C_HAL_Disable` (I2C\_Type \*base)  
*Disables the I2C module operation.*

## Pin functions

- static void `I2C_HAL_SetGlitchWidth` (I2C\_Type \*base, uint8\_t glitchWidth)  
*Controls the width of the programmable glitch filter.*

## Low power

- static void `I2C_HAL_SetWakeupCmd` (I2C\_Type \*base, bool enable)  
*Controls the I2C wakeup enable.*

## I2C HAL driver

### Baud rate

- void [I2C\\_HAL\\_SetBaudRate](#) (I2C\_Type \*base, uint32\_t sourceClockInHz, uint32\_t kbps, uint32\_t \*absoluteError\_Hz)  
*Sets the I2C bus frequency for master transactions.*
- static void [I2C\\_HAL\\_SetFreqDiv](#) (I2C\_Type \*base, uint8\_t mult, uint8\_t icr)  
*Sets the I2C baud rate multiplier and table entry.*
- static void [I2C\\_HAL\\_SetSlaveBaudCtrlCmd](#) (I2C\_Type \*base, bool enable)  
*Slave baud rate control.*

### Bus operations

- void [I2C\\_HAL\\_SendStart](#) (I2C\_Type \*base)  
*Sends a START or a Repeated START signal on the I2C bus.*
- [i2c\\_status\\_t](#) [I2C\\_HAL\\_SendStop](#) (I2C\_Type \*base)  
*Sends a STOP signal on the I2C bus.*
- static void [I2C\\_HAL\\_SendAck](#) (I2C\_Type \*base)  
*Causes an ACK to be sent on the bus.*
- static void [I2C\\_HAL\\_SendNak](#) (I2C\_Type \*base)  
*Causes a NAK to be sent on the bus.*
- static void [I2C\\_HAL\\_SetDirMode](#) (I2C\_Type \*base, [i2c\\_direction\\_t](#) direction)  
*Selects either transmit or receive mode.*
- static [i2c\\_direction\\_t](#) [I2C\\_HAL\\_GetDirMode](#) (I2C\_Type \*base)  
*Returns the currently selected transmit or receive mode.*

### Data transfer

- static uint8\_t [I2C\\_HAL\\_ReadByte](#) (I2C\_Type \*base)  
*Returns the last byte of data read from the bus and initiate another read.*
- static void [I2C\\_HAL\\_WriteByte](#) (I2C\_Type \*base, uint8\_t byte)  
*Writes one byte of data to the I2C bus.*
- uint8\_t [I2C\\_HAL\\_ReadByteBlocking](#) (I2C\_Type \*base)  
*Returns the last byte of data read from the bus and initiate another read.*
- bool [I2C\\_HAL\\_WriteByteBlocking](#) (I2C\_Type \*base, 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](#) (I2C\_Type \*base, 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](#) (I2C\_Type \*base, 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](#) (I2C\_Type \*base, const uint8\_t \*txBuff, uint32\_t txSize)  
*Send out multiple bytes of data using polling method.*
- [i2c\\_status\\_t](#) [I2C\\_HAL\\_SlaveReceiveDataPolling](#) (I2C\_Type \*base, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Receive multiple bytes of data using polling method.*



## Slave address

- void [I2C\\_HAL\\_SetAddress7bit](#) (I2C\_Type \*base, uint8\_t address)  
*Sets the primary 7-bit slave address.*
- void [I2C\\_HAL\\_SetAddress10bit](#) (I2C\_Type \*base, uint16\_t address)  
*Sets the primary slave address and enables 10-bit address mode.*
- static void [I2C\\_HAL\\_SetExtensionAddrCmd](#) (I2C\_Type \*base, bool enable)  
*Enables or disables the extension address (10-bit).*
- static bool [I2C\\_HAL\\_GetExtensionAddrCmd](#) (I2C\_Type \*base)  
*Returns whether the extension address is enabled or not.*
- static void [I2C\\_HAL\\_SetGeneralCallCmd](#) (I2C\_Type \*base, bool enable)  
*Controls whether the general call address is recognized.*
- static void [I2C\\_HAL\\_SetRangeMatchCmd](#) (I2C\_Type \*base, bool enable)  
*Enables or disables the slave address range matching.*
- static void [I2C\\_HAL\\_SetUpperAddress7bit](#) (I2C\_Type \*base, uint8\_t address)  
*Sets the upper slave address.*

## Status

- static bool [I2C\\_HAL\\_GetStatusFlag](#) (I2C\_Type \*base, [i2c\\_status\\_flag\\_t](#) statusFlag)  
*Gets the I2C status flag state.*
- static bool [I2C\\_HAL\\_IsMaster](#) (I2C\_Type \*base)  
*Returns whether the I2C module is in master mode.*
- static void [I2C\\_HAL\\_ClearArbitrationLost](#) (I2C\_Type \*base)  
*Clears the arbitration lost flag.*

## Interrupt

- static void [I2C\\_HAL\\_SetIntCmd](#) (I2C\_Type \*base, bool enable)  
*Enables or disables I2C interrupt requests.*
- static bool [I2C\\_HAL\\_GetIntCmd](#) (I2C\_Type \*base)  
*Returns whether the I2C interrupts are enabled.*
- static bool [I2C\\_HAL\\_IsIntPending](#) (I2C\_Type \*base)  
*Returns the current I2C interrupt flag.*
- static void [I2C\\_HAL\\_ClearInt](#) (I2C\_Type \*base)  
*Clears the I2C interrupt if set.*

## 21.2.4 Enumeration Type Documentation

### 21.2.4.1 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\_ArbitrationLost* 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.

### 21.2.4.2 enum i2c\_status\_flag\_t

### 21.2.4.3 enum i2c\_direction\_t

Enumerator

*kI2CReceive* Master transmit, slave receive.

*kI2CSend* Master receive, slave transmit.

## 21.2.5 Function Documentation

### 21.2.5.1 void I2C\_HAL\_Init ( I2C\_Type \* *base* )

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

### 21.2.5.2 static void I2C\_HAL\_Enable ( I2C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

### 21.2.5.3 static void I2C\_HAL\_Disable ( I2C\_Type \* *base* ) [inline], [static]

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

**21.2.5.4 static void I2C\_HAL\_SetGlitchWidth ( I2C\_Type \* *base*, 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>base</i>        | The I2C peripheral base pointer                                                                             |
| <i>glitchWidth</i> | Maximum width in bus clock cycles of the glitches that is filtered. Pass zero to disable the glitch filter. |

**21.2.5.5 static void I2C\_HAL\_SetWakeupCmd ( I2C\_Type \* *base*, 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>base</i>   | The I2C peripheral base pointer.                                                                                                                      |
| <i>enable</i> | true - Enables the wakeup function in low power mode.<br>false - Normal operation. No interrupt is generated when address matching in low power mode. |

**21.2.5.6 void I2C\_HAL\_SetBaudRate ( I2C\_Type \* *base*, uint32\_t *sourceClockInHz*,  
uint32\_t *kbps*, uint32\_t \* *absoluteError\_Hz* )**

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

## 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. |

### 21.2.5.7 static void I2C\_HAL\_SetFreqDiv ( I2C\_Type \* *base*, 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>base</i> | The I2C peripheral base pointer                                                                                                             |
| <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. |

### 21.2.5.8 static void I2C\_HAL\_SetSlaveBaudCtrlCmd ( I2C\_Type \* *base*, 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>base</i>   | The I2C peripheral base pointer                                                                                                                            |
| <i>enable</i> | true - Slave baud rate is independent of the master baud rate;<br>false - The slave baud rate follows the master baud rate and clock stretching may occur. |

### 21.2.5.9 void I2C\_HAL\_SendStart ( I2C\_Type \* *base* )

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>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

**21.2.5.10 i2c\_status\_t I2C\_HAL\_SendStop ( I2C\_Type \* *base* )**

This function changes the direction to receive.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

Returns

Whether the sending of STOP single is success or not.

**21.2.5.11 static void I2C\_HAL\_SendAck ( I2C\_Type \* *base* ) [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>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

**21.2.5.12 static void I2C\_HAL\_SendNak ( I2C\_Type \* *base* ) [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>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

**21.2.5.13** `static void I2C_HAL_SetDirMode ( I2C_Type * base, i2c_direction_t direction ) [inline], [static]`

### Parameters

|                  |                                                                                                                                                                                |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | The I2C peripheral base pointer.                                                                                                                                               |
| <i>direction</i> | Specifies either transmit mode or receive mode. The valid values are: <ul style="list-style-type: none"> <li>• #kI2CTransmit</li> <li>• <a href="#">kI2CReceive</a></li> </ul> |

**21.2.5.14** `static i2c_direction_t I2C_HAL_GetDirMode ( I2C_Type * base ) [inline], [static]`

### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base pointer. |
|-------------|----------------------------------|

### Returns

Current I2C transfer mode.

### Return values

|                               |                                                      |
|-------------------------------|------------------------------------------------------|
| <a href="#">#kI2CTransmit</a> | I2C is configured for master or slave transmit mode. |
| <a href="#">kI2CReceive</a>   | I2C is configured for master or slave receive mode.  |

**21.2.5.15** `static uint8_t I2C_HAL_ReadByte ( I2C_Type * base ) [inline], [static]`

In a master receive mode, calling this function initiates receiving the next byte of data.

### Parameters

---

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

**Returns**

This function returns the last byte received while the I2C module is configured in master receive or slave receive mode.

**21.2.5.16 static void I2C\_HAL\_WriteByte ( I2C\_Type \* *base*, 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>base</i> | The I2C peripheral base pointer. |
| <i>byte</i> | The byte of data to transmit.    |

**21.2.5.17 uint8\_t I2C\_HAL\_ReadByteBlocking ( I2C\_Type \* *base* )**

It will wait till the transfer is actually completed.

**Parameters**

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

**Returns**

Returns the last byte received

**21.2.5.18 bool I2C\_HAL\_WriteByteBlocking ( I2C\_Type \* *base*, uint8\_t *byte* )**

## I2C HAL driver

### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base pointer. |
| <i>byte</i> | The byte of data to transmit.    |

### Returns

Whether ACK is received(TRUE) or not(FALSE).

**21.2.5.19** `i2c_status_t I2C_HAL_MasterReceiveDataPolling ( I2C_Type * base, uint16_t slaveAddr, const uint8_t * cmdBuff, uint32_t cmdSize, uint8_t * rxBuff, uint32_t rxSize )`

### Parameters

|                  |                                                        |
|------------------|--------------------------------------------------------|
| <i>base</i>      | The I2C peripheral base pointer.                       |
| <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.

**21.2.5.20** `i2c_status_t I2C_HAL_MasterSendDataPolling ( I2C_Type * base, uint16_t slaveAddr, const uint8_t * cmdBuff, uint32_t cmdSize, const uint8_t * txBuff, uint32_t txSize )`

### Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>base</i>      | The I2C peripheral base pointer.  |
| <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.

#### 21.2.5.21 `i2c_status_t I2C_HAL_SlaveSendDataPolling ( I2C_Type * base, const uint8_t * txBuff, uint32_t txSize )`

Parameters

|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>base</i>   | I2C module base pointer.                            |
| <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

Whether the transaction is success or not.

Return values

|                                 |                                    |
|---------------------------------|------------------------------------|
| <i>kStatus_I2C_Received-Nak</i> | if received NACK bit               |
| <i>Error</i>                    | or success status returned by API. |

#### 21.2.5.22 `i2c_status_t I2C_HAL_SlaveReceiveDataPolling ( I2C_Type * base, uint8_t * rxBuff, uint32_t rxSize )`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | I2C module base pointer. |
|-------------|--------------------------|

## I2C HAL driver

|               |                                                         |
|---------------|---------------------------------------------------------|
| <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.

#### 21.2.5.23 void I2C\_HAL\_SetAddress7bit ( I2C\_Type \* *base*, uint8\_t *address* )

##### Parameters

|                |                                                                       |
|----------------|-----------------------------------------------------------------------|
| <i>base</i>    | The I2C peripheral base pointer                                       |
| <i>address</i> | The slave address in the upper 7 bits. Bit 0 of this value must be 0. |

#### 21.2.5.24 void I2C\_HAL\_SetAddress10bit ( I2C\_Type \* *base*, uint16\_t *address* )

##### Parameters

|                |                                                                         |
|----------------|-------------------------------------------------------------------------|
| <i>base</i>    | The I2C peripheral base pointer                                         |
| <i>address</i> | The 10-bit slave address, in bits [10:1] of the value. Bit 0 must be 0. |

#### 21.2.5.25 static void I2C\_HAL\_SetExtensionAddrCmd ( I2C\_Type \* *base*, bool *enable* ) [inline], [static]

##### Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | The I2C peripheral base pointer                                        |
| <i>enable</i> | true: 10-bit address is enabled. false: 10-bit address is not enabled. |

#### 21.2.5.26 static bool I2C\_HAL\_GetExtensionAddrCmd ( I2C\_Type \* *base* ) [inline], [static]

## Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

## Returns

true: 10-bit address is enabled. false: 10-bit address is not enabled.

**21.2.5.27 static void I2C\_HAL\_SetGeneralCallCmd ( I2C\_Type \* *base*, bool *enable* )**  
**[inline], [static]**

## Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | The I2C peripheral base pointer             |
| <i>enable</i> | Whether to enable the general call address. |

**21.2.5.28 static void I2C\_HAL\_SetRangeMatchCmd ( I2C\_Type \* *base*, bool *enable* )**  
**[inline], [static]**

## Parameters

|               |                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | The I2C peripheral base pointer.                                                                                                       |
| <i>enable</i> | Pass true to enable range address matching. You must also call <a href="#">I2C_HAL_SetUpperAddress7bit()</a> to set the upper address. |

**21.2.5.29 static void I2C\_HAL\_SetUpperAddress7bit ( I2C\_Type \* *base*, 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.

## I2C HAL driver

### Parameters

|                |                                                                                                                                                                                                                        |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>    | The I2C peripheral base pointer                                                                                                                                                                                        |
| <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 <a href="#">I2C_HAL_SetAddress7bit()</a> . |

### 21.2.5.30 `static bool I2C_HAL_GetStatusFlag ( I2C_Type * base, i2c_status_flag_t statusFlag ) [inline], [static]`

#### Parameters

|                   |                                                                   |
|-------------------|-------------------------------------------------------------------|
| <i>base</i>       | The I2C peripheral base pointer.                                  |
| <i>statusFlag</i> | The status flag, defined in type <code>i2c_status_flag_t</code> . |

#### 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.

### 21.2.5.31 `static bool I2C_HAL_IsMaster ( I2C_Type * base ) [inline], [static]`

#### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base pointer. |
|-------------|----------------------------------|

#### Returns

Whether current I2C is in master mode or not.

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

**21.2.5.32 static void I2C\_HAL\_ClearArbitrationLost ( I2C\_Type \* *base* ) [inline], [static]**

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

**21.2.5.33 static void I2C\_HAL\_SetIntCmd ( I2C\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>base</i>   | The I2C peripheral base pointer                  |
| <i>enable</i> | Pass true to enable interrupt, false to disable. |

**21.2.5.34 static bool I2C\_HAL\_GetIntCmd ( I2C\_Type \* *base* ) [inline], [static]**

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

Returns

Whether I2C interrupts are enabled or not.

**21.2.5.35 static bool I2C\_HAL\_IsIntPending ( I2C\_Type \* *base* ) [inline], [static]**

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

Returns

Whether I2C interrupt is pending or not.

**21.2.5.36 static void I2C\_HAL\_ClearInt ( I2C\_Type \* *base* ) [inline], [static]**

## I2C HAL driver

### Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | The I2C peripheral base pointer |
|-------------|---------------------------------|

## 21.3 I2C Slave peripheral driver

### 21.3.1 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](#) {  
[kI2CSlaveTxReq](#) = 0x02u,  
[kI2CSlaveRxReq](#) = 0x04u,  
[kI2CSlaveTxNAK](#) = 0x08u,  
[kI2CSlaveTxEmpty](#) = 0x10u,  
[kI2CSlaveRxFull](#) = 0x20u,  
[kI2CSlaveAbort](#) = 0x40u }  
*Internal driver state information.*

#### Variables

- I2C\_Type \*const [g\\_i2cBase](#) [I2C\_INSTANCE\_COUNT]  
*Table of base addresses for I2C instances.*

#### I2C Slave

- [i2c\\_status\\_t](#) [I2C\\_DRV\\_SlaveInit](#) (uint32\_t instance, const [i2c\\_slave\\_user\\_config\\_t](#) \*userConfigPtr, [i2c\\_slave\\_state\\_t](#) \*slave)  
*Initializes the I2C module.*
- [i2c\\_status\\_t](#) [I2C\\_DRV\\_SlaveDeinit](#) (uint32\_t instance)  
*Shuts down the I2C slave driver.*
- [i2c\\_slave\\_state\\_t](#) \* [I2C\\_DRV\\_SlaveGetHandler](#) (uint32\_t instance)

## I2C Slave peripheral driver

- Gets the 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 by 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 by using a blocking method.*
  - [i2c\\_status\\_t I2C\\_DRV\\_SlaveReceiveData](#) (uint32\_t instance, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Receives the data by using a non-blocking method.*
  - [i2c\\_status\\_t I2C\\_DRV\\_SlaveReceiveDataBlocking](#) (uint32\_t instance, uint8\_t \*rxBuff, uint32\_t rxSize, uint32\_t timeout\_ms)  
*Receives data by using a blocking method.*
  - [i2c\\_status\\_t I2C\\_DRV\\_SlaveGetReceiveStatus](#) (uint32\_t instance, uint32\_t \*bytesRemaining)  
*Gets the current status of the I2C slave driver.*
  - [i2c\\_status\\_t I2C\\_DRV\\_SlaveGetTransmitStatus](#) (uint32\_t instance, uint32\_t \*bytesRemaining)  
*Gets the current status of the I2C slave driver.*
  - [i2c\\_status\\_t I2C\\_DRV\\_SlaveAbortReceiveData](#) (uint32\_t instance, uint32\_t \*rxSize)  
*Terminates a non-blocking receive of the I2C slave early.*
  - [i2c\\_status\\_t I2C\\_DRV\\_SlaveAbortSendData](#) (uint32\_t instance, uint32\_t \*txSize)  
*Terminates a non-blocking send of the I2C slave early.*
  - static bool [I2C\\_DRV\\_SlaveIsBusBusy](#) (uint32\_t instance)  
*Gets the current status of the I2C slave bus.*
  - static [i2c\\_status\\_t I2C\\_DRV\\_SlaveSendDataPolling](#) (uint32\_t instance, const uint8\_t \*txBuff, uint32\_t txSize)  
*Sends out multiple bytes of data using a polling method.*
  - static [i2c\\_status\\_t I2C\\_DRV\\_SlaveReceiveDataPolling](#) (uint32\_t instance, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Receives multiple bytes of data using a polling method.*
  - void [I2C\\_DRV\\_SlaveIRQHandler](#) (uint32\_t instance)  
*The interrupt handler for I2C slave mode.*

### 21.3.2 Data Structure Documentation

#### 21.3.2.1 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.*
- 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.*

### 21.3.2.1.0.36 Field Documentation

**21.3.2.1.0.36.1** `i2c_status_t i2c_slave_state_t::status`

**21.3.2.1.0.36.2** `volatile uint32_t i2c_slave_state_t::txSize`

**21.3.2.1.0.36.3** `volatile uint32_t i2c_slave_state_t::rxSize`

**21.3.2.1.0.36.4** `const uint8_t* i2c_slave_state_t::txBuff`

**21.3.2.1.0.36.5** `uint8_t* i2c_slave_state_t::rxBuff`

**21.3.2.1.0.36.6** `bool i2c_slave_state_t::isTxBusy`

**21.3.2.1.0.36.7** `bool i2c_slave_state_t::isRxBusy`

**21.3.2.1.0.36.8** `bool i2c_slave_state_t::isTxBlocking`

**21.3.2.1.0.36.9** `bool i2c_slave_state_t::isRxBlocking`

**21.3.2.1.0.36.10** `bool i2c_slave_state_t::slaveListening`

**21.3.2.1.0.36.11** `i2c_slave_callback_t i2c_slave_state_t::slaveCallback`

**21.3.2.1.0.36.12** `void* i2c_slave_state_t::callbackParam`

**21.3.2.2** `struct i2c_slave_user_config_t`

## I2C Slave peripheral driver

### 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.*

### 21.3.2.2.0.37 Field Documentation

#### 21.3.2.2.0.37.1 uint16\_t i2c\_slave\_user\_config\_t::address

If 10-bit address, the first 6 bits must be 011110 in binary.

## 21.3.3 Enumeration Type Documentation

### 21.3.3.1 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

- kI2CSlaveTxReq*** The slave I2C Transmitting Request event.
- kI2CSlaveRxReq*** The slave I2C Receiving Request event.
- kI2CSlaveTxNAK*** The slave I2C Transmitting NAK event.
- kI2CSlaveTxEmpty*** The slave I2C Transmitting Buffer Empty event.
- kI2CSlaveRxFull*** The slave I2C Receiving Buffer Full event.
- kI2CSlaveAbort*** The slave I2C Slave abort transaction event.

### 21.3.4 Function Documentation

**21.3.4.1** `i2c_status_t 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.

## I2C Slave peripheral driver

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.    |

Returns

Error or success status returned by API.

### 21.3.4.2 `i2c_status_t 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. |
|-----------------|------------------------------------|

Returns

Error or success status returned by API.

### 21.3.4.3 `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.

### 21.3.4.4 `i2c_status_t I2C_DRV_SlaveSendData ( uint32_t instance, const uint8_t * txBuff, uint32_t txSize )`

This function returns immediately when the buffer pointer and length are set to the transfer buffer and transfer size. The user should check the status of I2C slave to find out whether the transmission is completed. The user can also wait the `kI2CSlaveStop` or the `kI2CSlaveTxDone` to ensure that the transmission is ended.

### Parameters

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>instance</i> | Instance number of the I2C module.                |
| <i>txBuff</i>   | The pointer to sending the data buffer.           |
| <i>txSize</i>   | The number of bytes which the user wants to send. |

### Returns

success (if I2C slave status is not error) or error code in others.

#### 21.3.4.5 **i2c\_status\_t I2C\_DRV\_SlaveSendDataBlocking ( uint32\_t instance, const uint8\_t \* txBuff, uint32\_t txSize, uint32\_t timeout\_ms )**

This function sets the buffer pointer and length to the transfer buffer and the transfer size and waits until the transmission is ended (NAK is detected).

### Parameters

|                   |                                                                   |
|-------------------|-------------------------------------------------------------------|
| <i>instance</i>   | Instance number of the I2C module.                                |
| <i>txBuff</i>     | The pointer to sending the data buffer.                           |
| <i>txSize</i>     | The number of bytes which the 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.

#### 21.3.4.6 **i2c\_status\_t I2C\_DRV\_SlaveReceiveData ( uint32\_t instance, uint8\_t \* rxBuff, uint32\_t rxSize )**

This function returns immediately when the buffer pointer and length are set to the receive buffer and the receive size. The user should check the status of the I2C slave to find out whether the transmission is completed. The user can also wait the kI2CSlaveStop or the kI2CSlaveRxDone to ensure that the transmission is ended.

### Parameters

---

## I2C Slave peripheral driver

|                 |                                                      |
|-----------------|------------------------------------------------------|
| <i>instance</i> | Instance number of the I2C module.                   |
| <i>rxBuff</i>   | The pointer to the received data buffer.             |
| <i>rxSize</i>   | The number of bytes which the user wants to receive. |

### Returns

success (if I2C slave status is not error) or error code in others.

### 21.3.4.7 `i2c_status_t I2C_DRV_SlaveReceiveDataBlocking ( uint32_t instance, uint8_t * rxBuff, uint32_t rxSize, uint32_t timeout_ms )`

This function sets the buffer pointer and length to the receive buffer and the receive size. Then, the function waits until the transmission is ended (all data is received or a STOP signal is detected).

### Parameters

|                   |                                                                  |
|-------------------|------------------------------------------------------------------|
| <i>instance</i>   | Instance number of the I2C module.                               |
| <i>rxBuff</i>     | The pointer to the received data buffer.                         |
| <i>rxSize</i>     | The number of bytes which the 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.

### 21.3.4.8 `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).

### 21.3.4.9 `i2c_status_t I2C_DRV_SlaveGetTransmitStatus ( 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).

#### 21.3.4.10 `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.

#### 21.3.4.11 `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.

#### 21.3.4.12 `static bool I2C_DRV_SlavelBusBusy ( uint32_t instance ) [inline], [static]`

## I2C Slave peripheral driver

### Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>instance</i> | Instance number of the I2C module. |
|-----------------|------------------------------------|

### Returns

True if the bus is busy False if the bus is idle

**21.3.4.13** `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 bytes.                   |

### Returns

Error or success status returned by API.

**21.3.4.14** `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 bytes.              |

### Returns

Error or success status returned by the API.

**21.3.4.15** `void I2C_DRV_SlaveIRQHandler ( uint32_t instance )`



Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>instance</i> | Instance number of the I2C module. |
|-----------------|------------------------------------|

### 21.3.5 Variable Documentation

#### 21.3.5.1 I2C\_Type\* const g\_i2cBase[I2C\_INSTANCE\_COUNT]

## I2C Master peripheral

### 21.4 I2C Master peripheral

#### 21.4.1 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.

#### 21.4.2 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` from OSA layer, `OSA_Init` must be called before calling I2C transaction API functions. Otherwise, the API functions do not work.

#### 21.4.3 I2C Data Transactions

I2C master driver provides these 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

- I2C\_Type \*const [g\\_i2cBase](#) [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.*
- [i2c\\_status\\_t 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 the current status of the 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 the current status of the I2C master receive.*
- 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.*
- void [I2C\\_DRV\\_MasterIRQHandler](#) (uint32\_t instance)  
*The interrupt handler for I2C master mode.*

## I2C Master peripheral

### 21.4.4 Data Structure Documentation

#### 21.4.4.1 struct i2c\_device\_t

##### Data Fields

- uint16\_t [address](#)  
*Slave's 7-bit or 10-bit address.*
- uint32\_t [baudRate\\_kbps](#)  
*The baud rate in kbps to use by current slave device.*

##### 21.4.4.1.0.38 Field Documentation

###### 21.4.4.1.0.38.1 uint16\_t i2c\_device\_t::address

If 10-bit address, the first 6 bits must be 011110 in binary.

#### 21.4.4.2 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.

### 21.4.5 Function Documentation

#### 21.4.5.1 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.

#### 21.4.5.2 i2c\_status\_t I2C\_DRV\_MasterDeinit ( uint32\_t *instance* )

### Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The I2C peripheral instance number. |
|-----------------|-------------------------------------|

### Returns

Error or success status returned by API.

#### 21.4.5.3 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. |

#### 21.4.5.4 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.

## I2C Master peripheral

**21.4.5.5** `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 transfer buffer and transfer 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

|                 |                                                                    |
|-----------------|--------------------------------------------------------------------|
| <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.

**21.4.5.6** `i2c_status_t I2C_DRV_MasterGetSendStatus ( uint32_t instance, uint32_t * bytesRemaining )`

This function gets the current I2C status of the 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).

**21.4.5.7** `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.

**21.4.5.8** `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.

### 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.

**21.4.5.9** `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 the receive buffer and the receive 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.

## 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.    |

### Returns

Error or success status returned by API.

#### 21.4.5.10 **i2c\_status\_t I2C\_DRV\_MasterGetReceiveStatus ( uint32\_t *instance*, uint32\_t \* *bytesRemaining* )**

This function is designed to get the current I2C status of a 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).

#### 21.4.5.11 **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. The user needs to prepare these buffers according to the related protocol if the slave device data is 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.

**21.4.5.12** `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. The user needs to prepare these buffers according to the related protocol if the slave device data is 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>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.

**21.4.5.13** `void I2C_DRV_MasterIRQHandler ( uint32_t instance )`

## I2C Master peripheral

Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>instance</i> | Instance number of the I2C module. |
|-----------------|------------------------------------|

### 21.4.6 Variable Documentation

#### 21.4.6.1 I2C\_Type\* const g\_i2cBase[I2C\_INSTANCE\_COUNT]

## Chapter 22

# Independent Real Time Clock (IRTC)

### 22.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Independent Real Time Clock (IRTC) block of Kinetis devices.

#### Modules

- [IRTC HAL driver](#)
- [IRTC Peripheral Driver](#)

## IRTC HAL driver

### 22.2 IRTC HAL driver

This section describes the programming interface of the IRTC HAL driver. The IRTC HAL driver initializes the IRTC registers and provides functions to read or modify the IRTC registers. These are mostly invoked by the IRTC Peripheral driver.

#### 22.2.1 IRTC Initialization

This function initiates a soft-reset of the IRTC module to reset the IRTC registers and configure IRTC according to user settings. It doesn't un-gate the IRTC clock.

#### 22.2.2 IRTC Setting and reading the IRTC time

The HAL driver provides [IRTC\\_HAL\\_SetDatetime\(\)](#) and [IRTC\\_HAL\\_GetDatetime\(\)](#) functions to set and read the date and time using an instantiation of the `irtc_datetime_t` structure. This example describes the structure:

```
typedef struct IrtcDatetime
{
 uint16_t year;
 uint16_t month;
 uint16_t day;
 uint16_t weekDay;
 uint16_t hour;
 uint16_t minute;
 uint16_t second;
} irtc_datetime_t;
```

#### 22.2.3 IRTC Setting and reading the Alarm

The HAL driver provides [IRTC\\_HAL\\_SetAlarm\(\)](#) and [IRTC\\_HAL\\_GetAlarm\(\)](#) functions to set an alarm and read back the alarm time.

## 22.3 IRTC Peripheral Driver

### 22.3.1 Overview

This section describes the programming interface of the IRTC Peripheral Driver. The IRTC Peripheral driver sets and gets the IRTC clock, sets and reads the IRTC alarm time.

### 22.3.2 IRTC Peripheral Driver Initialization

To initialize, call the `IRTC_DRV_Init()` function with the IRTC instance number. Most SoCs have only one IRTC instance. Therefore, the instance number is zero. The driver initialization function un-gates the IRTC module clock, initializes the IRTC HAL layer driver, and enables the IRTC interrupts when an alarm is set.

This is an example how to create the `irtc_init()` function:

```
// init the irtc module //
IRTC_DRV_Init(0);
```

### 22.3.3 IRTCS Setting and reading the IRTC time

The IRTC driver uses an instantiation of the `irtc_datetime_t` structure either to configure or read the date and time. Call the `IRTC_DRV_SetDatetime()` function to configure the current date and time and call the `IRTC_DRV_GetDatetime()` function to read the current date and time at a later time. This example shows how to use these functions:

```
irtc_datetime_t datetimeToSet;

IRTC_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 = IRTC_DRV_SetDatetime(0, &datetime);

// get datetime //
IRTC_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);
```

### 22.3.4 IRTC Triggering an Alarm

Call the `IRTC_DRV_SetAlarm()` function to set the alarm time and call the `IRTC_DRV_GetAlarm()` function to read the configured alarm time. Set the current time using the steps mentioned earlier prior

## IRTC Peripheral Driver

to using 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 calling the `IRTC_DRV_SetIntCmd()` and `IRTC_DRV_SetAlarmMatchMode()` functions. This is an example to set an IRTC alarm. This example causes an alarm interrupt to be triggered after 5 minutes.

```

irtc_datetime_t datetimeToSet;
irtc_datetime_t alrmTimeToSet;

IRTC_DRV_Init(0);

datetimeToSet.year = 2013;
datetimeToSet.month = 10;
datetimeToSet.day = 13;
datetimeToSet.hour = 18;
datetimeToSet.minute = 55;
datetimeToSet.second = 30;

IRTC_DRV_SetDatetime(0, &datetimeToSet);

alrmTimeToSet.year = datetimeToSet.year;
alrmTimeToSet.month = datetimeToSet.month;
alrmTimeToSet.day = datetimeToSet.day;
alrmTimeToSet.minute = datetimeToSet.minute + 5;
alrmTimeToSet.second = datetimeToSet.second;
IRTC_DRV_SetAlarm(0, &alrmTimeToSet);
IRTC_DRV_SetAlarmMatchMode(0,
 kIRTCSecMinHourDayMonYear);
IRTC_HAL_SetIntCmd(0, kIRTCAlarmIntFlag);

```

## Variables

- RTC\_Type \*const `g_irtcBase` [RTC\_INSTANCE\_COUNT]  
*Table of base addresses for IRTC instances.*
- const IRQn\_Type `g_irtcIrqId` [RTC\_INSTANCE\_COUNT]  
*Table to save IRTC Alarm IRQ numbers for IRTC instances.*

## Initialization and De-initialization

- void `IRTC_DRV_Init` (uint32\_t instance, const `irtc_datetime_t` \*datetime, const `irtc_datetime_t` \*alarmDatetime, `irtc_alarm_match_t` alarmMode, const `irtc_daylight_time_t` \*daylightTime)  
*Initializes the IRTC module.*
- void `IRTC_DRV_Deinit` (uint32\_t instance)  
*Disables the IRTC module clock gate control.*

## IRTC Datetime Set and Get

- static void `IRTC_DRV_SetDatetime` (uint32\_t instance, `irtc_datetime_t` \*datetime)  
*Sets the IRTC date and time according to the given time structure.*
- static void `IRTC_DRV_GetDatetime` (uint32\_t instance, `irtc_datetime_t` \*datetime)  
*Gets the IRTC time and stores it in the given time structure.*

### IRTC Alarm

- static void [IRTC\\_DRV\\_SetAlarmMatchMode](#) (uint32\_t instance, [irtc\\_alarm\\_match\\_t](#) alarmType)  
*Sets alarm match type.*
- static void [IRTC\\_DRV\\_SetAlarm](#) (uint32\_t instance, [irtc\\_datetime\\_t](#) \*alarmTime)  
*Sets the IRTC alarm time.*
- static void [IRTC\\_DRV\\_GetAlarm](#) (uint32\_t instance, [irtc\\_datetime\\_t](#) \*date)  
*Returns the IRTC alarm time.*

### Interrupts

- static void [IRTC\\_DRV\\_SetIntCmd](#) (uint32\_t instance, [irtc\\_int\\_t](#) interrupt, bool enable)  
*Enables or disables the related IRTC interrupt.*
- static bool [IRTC\\_DRV\\_GetIntCmd](#) (uint32\_t instance, [irtc\\_int\\_t](#) interrupt)  
*Gets whether the IRTC interrupt is enabled or not.*
- static bool [IRTC\\_DRV\\_GetIntStatusFlag](#) (uint32\_t instance, [irtc\\_int\\_status\\_flag\\_t](#) statusFlag)  
*Gets the IRTC interrupt status flag state.*
- static void [IRTC\\_DRV\\_ClearIntStatusFlag](#) (uint32\_t instance, [irtc\\_int\\_status\\_flag\\_t](#) statusFlag)  
*Clears the IRTC interrupt status flag.*

## 22.3.5 Function Documentation

### 22.3.5.1 void IRTC\_DRV\_Init ( uint32\_t instance, const irtc\_datetime\_t \* datetime, const irtc\_datetime\_t \* alarmDatetime, irtc\_alarm\_match\_t alarmMode, const irtc\_daylight\_time\_t \* daylightTime )

Enables the IRTC clock and interrupts if requested by the user.

Parameters

|                      |                                                          |
|----------------------|----------------------------------------------------------|
| <i>instance</i>      | The IRTC peripheral instance number.                     |
| <i>datetime</i>      | Date and time need to set, pass NULL to ignore.          |
| <i>alarmDatetime</i> | Alarm of date and time need to set, pass NULL to ignore. |
| <i>alarmMode</i>     | Alarm mode to set when will generate alarm.              |
| <i>daylightTime</i>  | Daylight saving time need to set, pass NULL to ignore.   |

### 22.3.5.2 void IRTC\_DRV\_Deinit ( uint32\_t instance )

## IRTC Peripheral Driver

Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>instance</i> | The IRTC peripheral instance number. |
|-----------------|--------------------------------------|

**22.3.5.3** `static void IRTC_DRV_SetDatetime ( uint32_t instance, irtc_datetime_t * datetime ) [inline], [static]`

Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>instance</i> | The IRTC peripheral instance number.                                    |
| <i>datetime</i> | Pointer to structure where the date and time details to set are stored. |

**22.3.5.4** `static void IRTC_DRV_GetDatetime ( uint32_t instance, irtc_datetime_t * datetime ) [inline], [static]`

Parameters

|                 |                                                                  |
|-----------------|------------------------------------------------------------------|
| <i>instance</i> | The IRTC peripheral instance number.                             |
| <i>datetime</i> | Pointer to structure where the date and time details are stored. |

**22.3.5.5** `static void IRTC_DRV_SetAlarmMatchMode ( uint32_t instance, irtc_alarm_match_t alarmType ) [inline], [static]`

Parameters

|                  |                                                        |
|------------------|--------------------------------------------------------|
| <i>instance</i>  | The IRTC peripheral instance number.                   |
| <i>alarmType</i> | Alarm match selections that when an alarm will happen. |

**22.3.5.6** `static void IRTC_DRV_SetAlarm ( uint32_t instance, irtc_datetime_t * alarmTime ) [inline], [static]`

Parameters

---



|                  |                                                     |
|------------------|-----------------------------------------------------|
| <i>instance</i>  | The IRTC peripheral instance number.                |
| <i>alarmTime</i> | Pointer to structure where the alarm time is store. |

**22.3.5.7 static void IRTC\_DRV\_GetAlarm ( uint32\_t *instance*, irtc\_datetime\_t \* *date* ) [inline], [static]**

Parameters

|                 |                                                                        |
|-----------------|------------------------------------------------------------------------|
| <i>instance</i> | The IRTC peripheral instance number.                                   |
| <i>date</i>     | Pointer to structure where the alarm date and time details are stored. |

**22.3.5.8 static void IRTC\_DRV\_SetIntCmd ( uint32\_t *instance*, irtc\_int\_t *interrupt*, bool *enable* ) [inline], [static]**

Parameters

|                  |                                                   |
|------------------|---------------------------------------------------|
| <i>instance</i>  | The IRTC peripheral instance number.              |
| <i>interrupt</i> | The interrupt name, defined in type irtc_int_t.   |
| <i>enable</i>    | Enable(true) or disable(false) related interrupt. |

**22.3.5.9 static bool IRTC\_DRV\_GetIntCmd ( uint32\_t *instance*, irtc\_int\_t *interrupt* ) [inline], [static]**

Parameters

|                  |                                                 |
|------------------|-------------------------------------------------|
| <i>instance</i>  | The IRTC peripheral instance number.            |
| <i>interrupt</i> | The interrupt name, defined in type irtc_int_t. |

Returns

State of the interrupt: asserted (true) or not-asserted (false).

- true: related interrupt is being enabled.
- false: related interrupt is not enabled.

**22.3.5.10 static bool IRTC\_DRV\_GetIntStatusFlag ( uint32\_t *instance*, irtc\_int\_status\_flag\_t *statusFlag* ) [inline], [static]**

## IRTC Peripheral Driver

### Parameters

|                   |                                                                        |
|-------------------|------------------------------------------------------------------------|
| <i>instance</i>   | The IRTC peripheral instance number.                                   |
| <i>statusFlag</i> | The status flag, defined in type <code>irtc_int_status_flag_t</code> . |

### 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.

**22.3.5.11** `static void IRTC_DRV_ClearIntStatusFlag ( uint32_t instance, irtc_int_status_flag_t statusFlag ) [inline], [static]`

Tamper interrupt status flag is cleared when the TAMPER\_SCR[TMPR\_STS] is cleared.

### Parameters

|                   |                                                                        |
|-------------------|------------------------------------------------------------------------|
| <i>instance</i>   | The IRTC peripheral instance number.                                   |
| <i>statusFlag</i> | The status flag, defined in type <code>irtc_int_status_flag_t</code> . |

## 22.3.6 Variable Documentation

**22.3.6.1** `RTC_Type* const g_irtcBase[RTC_INSTANCE_COUNT]`

**22.3.6.2** `const IRQn_Type g_irtclrqld[RTC_INSTANCE_COUNT]`

## Chapter 23

# Local Memory Controller (LMEM) Cache Control Driver

### 23.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Local Memory Controller Cache Controller. The Kinetis devices contain a Processor Code (PC) bus and a Processor System (PS) bus. Depending on the specific Kinetis MCU device, there may be a cache controller for both the PC bus and PS bus or there may only be one cache controller for the PC bus. Refer to the Kinetis reference manual to ascertain the availability of the cache. The LMEM Cache driver allows the user to enable/disable the cache and to perform cache maintenance operations such as invalidate, push, and clear. These maintenance operations may be performed on the entire cache or on a line-basis.

### Modules

- [LMEM Cache Driver](#)
- [LMEM Cache HAL driver](#)

## LMEM Cache HAL driver

### 23.2 LMEM Cache HAL driver

#### 23.2.1 Overview

This section describes the programming interface of the LMEM Cache HAL driver.

#### Files

- file [fsl\\_lmem\\_cache\\_hal.h](#)

#### Macros

- #define [LMEM\\_CACHE\\_LINE\\_SIZE](#) 0x10  
*LMEM CACHE Line Size in bytes.*

#### Enumerations

- enum [lmem\\_cache\\_status\\_t](#) { ,  
[kStatus\\_LMEM\\_CACHE\\_Busy](#),  
[kStatus\\_LMEM\\_CACHE\\_DemoteError](#),  
[kStatus\\_LMEM\\_CACHE\\_Error](#) }  
*Error codes for the LMEM CACHE driver.*
- enum [lmem\\_cache\\_mode\\_t](#) {  
[kCacheNonCacheable](#) = 0x0U,  
[kCacheWriteThrough](#) = 0x2U,  
[kCacheWriteBack](#) = 0x3U }  
*LMEM CACHE mode options.*
- enum [lmem\\_cache\\_region\\_t](#) {  
[kCacheRegion0](#) = 0U,  
[kCacheRegion1](#) = 1U,  
[kCacheRegion2](#) = 2U,  
[kCacheRegion3](#) = 3U,  
[kCacheRegion4](#) = 4U,  
[kCacheRegion5](#) = 5U,  
[kCacheRegion6](#) = 6U,  
[kCacheRegion7](#) = 7U,  
[kCacheRegion8](#) = 8U,  
[kCacheRegion9](#) = 9U,  
[kCacheRegion10](#) = 10U,  
[kCacheRegion11](#) = 11U,  
[kCacheRegion12](#) = 12U,  
[kCacheRegion13](#) = 13U,  
[kCacheRegion14](#) = 14U,  
[kCacheRegion15](#) = 15U }

*LMEM CACHE Regions.*

- enum `lmem_cache_line_command_t` {  
`kCacheLineSearchReadOrWrite` = 0U,  
`kCacheLineInvalidate` = 1U,  
`kCacheLinePush` = 2U,  
`kCacheLineClear` = 3U }

*LMEM CACHE line command.*

## Processor Code Bus Cache Control

- static void `LMEM_HAL_SetCodeCacheEnableCmd` (LMEM\_Type \*base, bool enable)  
*Enables or disables the Processor Code bus cache and write buffer.*
- void `LMEM_HAL_SetCodeCacheInvalidateAllCmd` (LMEM\_Type \*base, bool enable)  
*Enable or disable the Processor Code bus option to invalidate all lines.*
- void `LMEM_HAL_SetCodeCachePushAllCmd` (LMEM\_Type \*base, bool enable)  
*Enable or disable the Processor Code bus option to push all modified lines.*
- void `LMEM_HAL_SetCodeCacheClearAllCmd` (LMEM\_Type \*base, bool enable)  
*Enable or disable the Processor Code bus option to push and invalidate all modified lines.*
- static void `LMEM_HAL_InitiateCodeCacheCommand` (LMEM\_Type \*base)  
*Initiate the Processor Code bus cache command.*
- static bool `LMEM_HAL_IsCodeCacheCommandActive` (LMEM\_Type \*base)  
*Returns whether or not the Processor Code bus cache command is in progress.*
- static void `LMEM_HAL_InitiateCodeCacheLineCommand` (LMEM\_Type \*base)  
*Initiate the Processor Code bus cache line command.*
- static bool `LMEM_HAL_IsCodeCacheLineCommandActive` (LMEM\_Type \*base)  
*Returns whether or not the Processor Code bus cache line command is in progress.*
- static void `LMEM_HAL_SetCodeCacheLineCommand` (LMEM\_Type \*base, `lmem_cache_line_command_t` command)  
*Sets the cache line command for the Processor Code bus.*
- static void `LMEM_HAL_SetCodeCachePhysicalAddr` (LMEM\_Type \*base, uint32\_t addr)  
*Sets the physical address for cache line commands for the Processor Code bus.*
- void `LMEM_HAL_SetCodeCacheRegionMode` (LMEM\_Type \*base, `lmem_cache_region_t` region, `lmem_cache_mode_t` cacheMode)  
*Sets the cache mode for a specific region for the Processor Code bus.*
- uint32\_t `LMEM_HAL_GetCodeCacheRegionMode` (LMEM\_Type \*base, `lmem_cache_region_t` region)  
*Gets the current cache mode for a specific region for the Processor Code bus.*

## 23.2.2 Macro Definition Documentation

### 23.2.2.1 #define LMEM\_CACHE\_LINE\_SIZE 0x10

Cache line is 32 bytes (or 4-words)

## LMEM Cache HAL driver

### 23.2.3 Enumeration Type Documentation

#### 23.2.3.1 enum lmem\_cache\_status\_t

Enumerator

*kStatus\_LMEM\_CACHE\_Busy* CACHE busy performing an operation.  
*kStatus\_LMEM\_CACHE\_DemoteError* CACHE region demotion error.  
*kStatus\_LMEM\_CACHE\_Error* CACHE driver error.

#### 23.2.3.2 enum lmem\_cache\_mode\_t

Enumerator

*kCacheNonCacheable* CACHE mode: non-cacheable.  
*kCacheWriteThrough* CACHE mode: write-through.  
*kCacheWriteBack* CACHE mode: write-back.

#### 23.2.3.3 enum lmem\_cache\_region\_t

Enumerator

*kCacheRegion0* Cache Region 0.  
*kCacheRegion1* Cache Region 1.  
*kCacheRegion2* Cache Region 2.  
*kCacheRegion3* Cache Region 3.  
*kCacheRegion4* Cache Region 4.  
*kCacheRegion5* Cache Region 5.  
*kCacheRegion6* Cache Region 6.  
*kCacheRegion7* Cache Region 7.  
*kCacheRegion8* Cache Region 8.  
*kCacheRegion9* Cache Region 9.  
*kCacheRegion10* Cache Region 10.  
*kCacheRegion11* Cache Region 11.  
*kCacheRegion12* Cache Region 12.  
*kCacheRegion13* Cache Region 13.  
*kCacheRegion14* Cache Region 14.  
*kCacheRegion15* Cache Region 15.

#### 23.2.3.4 enum lmem\_cache\_line\_command\_t

Enumerator

*kCacheLineSearchReadOrWrite* Cache line search and read or write.

*kCacheLineInvalidate* Cache line invalidate.  
*kCacheLinePush* Cache line push.  
*kCacheLineClear* Cache line clear.

### 23.2.4 Function Documentation

#### 23.2.4.1 static void LMEM\_HAL\_SetCodeCacheEnableCmd ( LMEM\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables or disables the Processor Code bus cache and write buffer.

Parameters

|               |                                                                                |
|---------------|--------------------------------------------------------------------------------|
| <i>base</i>   | Module base pointer of type LMEM_Type.                                         |
| <i>enable</i> | Enable (true) or disable (false) the Processor Code bus cache and write buffer |

#### 23.2.4.2 void LMEM\_HAL\_SetCodeCacheInvalidateAllCmd ( LMEM\_Type \* *base*, bool *enable* )

This function enables or disables the Processor Code bus option to invalidate all lines in both WAYs.

Parameters

|               |                                                                                        |
|---------------|----------------------------------------------------------------------------------------|
| <i>base</i>   | Module base pointer of type LMEM_Type.                                                 |
| <i>enable</i> | Enable (true) or disable (false) the Processor Code bus option to invalidate all lines |

#### 23.2.4.3 void LMEM\_HAL\_SetCodeCachePushAllCmd ( LMEM\_Type \* *base*, bool *enable* )

This function enables or disables the Processor Code bus option to push all modified lines to both WAYs.

Parameters

|               |                                                                                           |
|---------------|-------------------------------------------------------------------------------------------|
| <i>base</i>   | Module base pointer of type LMEM_Type.                                                    |
| <i>enable</i> | Enable (true) or disable (false) the Processor Code bus option to push all modified lines |

#### 23.2.4.4 void LMEM\_HAL\_SetCodeCacheClearAllCmd ( LMEM\_Type \* *base*, bool *enable* )

This function enables or disables the Processor Code bus option to push and invalidate all modified lines.

## LMEM Cache HAL driver

### Parameters

|               |                                                                                           |
|---------------|-------------------------------------------------------------------------------------------|
| <i>base</i>   | Module base pointer of type LMEM_Type.                                                    |
| <i>enable</i> | Enable (true) or disable (false) the Processor Code bus option to push all modified lines |

### 23.2.4.5 static void LMEM\_HAL\_InitiateCodeCacheCommand ( LMEM\_Type \* *base* ) [inline], [static]

This function initiates the Processor Code bus cache command to execute an invalidate command and/or push command.

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Module base pointer of type LMEM_Type. |
|-------------|----------------------------------------|

### 23.2.4.6 static bool LMEM\_HAL\_IsCodeCacheCommandActive ( LMEM\_Type \* *base* ) [inline], [static]

This function returns the state of the Processor Code bus cache command. The command is either active (in progress) or idle.

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Module base pointer of type LMEM_Type. |
|-------------|----------------------------------------|

### Returns

True if the cache command is in progress or false if the command is idle

### 23.2.4.7 static void LMEM\_HAL\_InitiateCodeCacheLineCommand ( LMEM\_Type \* *base* ) [inline], [static]

This function initiates the Processor Code bus cache line command to execute a search and read or write command, an invalidate command, a push command, or a clear command.

### Parameters

---



|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Module base pointer of type LMEM_Type. |
|-------------|----------------------------------------|

**23.2.4.8 static bool LMEM\_HAL\_IsCodeCacheLineCommandActive ( LMEM\_Type \* *base* ) [inline], [static]**

This function returns the state of the Processor Code bus cache line command. The command is either active (in progress) or idle.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Module base pointer of type LMEM_Type. |
|-------------|----------------------------------------|

Returns

True if the cache line command is in progress or false if the command is idle

**23.2.4.9 static void LMEM\_HAL\_SetCodeCacheLineCommand ( LMEM\_Type \* *base*, lmem\_cache\_line\_command\_t *command* ) [inline], [static]**

This function sets the cache line command for the Processor Code bus. The command can be search and read or write, invalidate, push, or clear.

Parameters

|                |                                                          |
|----------------|----------------------------------------------------------|
| <i>base</i>    | Module base pointer of type LMEM_Type.                   |
| <i>command</i> | The cache line command of type lmem_cache_line_command_t |

**23.2.4.10 static void LMEM\_HAL\_SetCodeCachePhysicalAddr ( LMEM\_Type \* *base*, uint32\_t *addr* ) [inline], [static]**

This function sets the physical address for cache line commands for the Processor Code bus. The commands are specified in the CLCR[LADSEL] bits.

Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <i>base</i> | Module base pointer of type LMEM_Type.       |
| <i>addr</i> | The physical address for cache line commands |

## LMEM Cache HAL driver

**23.2.4.11 void LMEM\_HAL\_SetCodeCacheRegionMode ( LMEM\_Type \* *base*,  
lmem\_cache\_region\_t *region*, lmem\_cache\_mode\_t *cacheMode* )**

This function sets the cache mode for a specific region for the Processor Code bus. Note that you can only demote the cache mode.

Parameters

|                  |                                                                 |
|------------------|-----------------------------------------------------------------|
| <i>base</i>      | Module base pointer of type LMEM_Type.                          |
| <i>region</i>    | The region to demote the cache mode of type lmem_cache_region_t |
| <i>cacheMode</i> | The specified demoted cache mode of type lmem_cache_mode_t      |

**23.2.4.12 uint32\_t LMEM\_HAL\_GetCodeCacheRegionMode ( LMEM\_Type \* *base*, lmem\_cache\_region\_t *region* )**

This function gets the current cache mode for a specific region for the Processor Code bus.

Parameters

|               |                                                                 |
|---------------|-----------------------------------------------------------------|
| <i>base</i>   | Module base pointer of type LMEM_Type.                          |
| <i>region</i> | The region to obtain the cache mode of type lmem_cache_region_t |

Returns

The current cache mode for the specified region

## LMEM Cache Driver

### 23.3 LMEM Cache Driver

#### 23.3.1 Overview

This section describes the programming interface of the LMEM Cache peripheral driver. The LMEM Cache peripheral driver allows the user to enable/disable the cache and to perform cache maintenance operations such as invalidate, push, and clear. These maintenance operations may be performed on the entire cache or on a line-basis. In addition, the the driver also allows the user to "demote" the cache mode of a region within the device's memory map.

#### 23.3.2 LMEM Cache Driver Definitions and Usage

The following provides definitions to the terms commonly used in the LMEM Cache peripheral driver and how to use the various functions.

The Cortex-M4 processor has a modified 32-bit Harvard bus architecture. Processor Code (PC) bus - a 32-bit address space bus with low-order addresses (0x0000\_0000 through 0x1FFF\_FFFF) used normally for code access.

Processor System (PS) bus - a 32-bit address space bus with high-order addresses (0x2000\_0000 through 0xFFFF\_FFFF) used normally for data accesses.

Some Kinetic MCU devices have caches available for the PC bus and PS bus, some may only have a PC bus cache, while some do not have PC or PS caches at all. Refer to the desired Kinetic reference manual for cache availability.

Cache maintenance operations: Invalidate - Unconditionally clear valid and modify bits of a cache entry.

Push - Push a cache entry if it is valid and modified, then clear the modify bit. If entry not valid or not modified, leave as is. This action does not clear the valid bit. A cache push is synonymous with a cache flush.

Clear - Push a cache entry if it is valid and modified, then clear the valid and modify bits. If entry not valid or not modified, clear the valid bit.

The above cache maintenance operations may be performed on the entire cache or on a line-basis. The peripheral driver API names distinguish between the two using the terms "All" or "Line". For example, to perform an invalidate all on the PC bus, the API is called:

```
LMEM_DRV_CodeCacheInvalidateAll(uint32_t instance);
```

To invalidate a particular line, simple call:

```
LMEM_DRV_CodeCacheInvalidateLine(uint32_t instance, uint32_t addr);
```

Note that the parameter "addr" must be supplied which indicates the physical address of the line you wish to perform the cache maintenance operation.

In addition, if the user wishes to perform cache maintenance operations on multiple lines, there are APIs available which use the naming convention "MultiLines". For example, to perform a multi-line push on the PC bus, the API is called:

```
LMEM_DRV_CodeCachePushMultiLines(uint32_t instance, uint32_t addr, uint32_t
 length);
```

Note that the parameter "addr" must be supplied which indicates the starting physical address of the lines you wish to perform the cache maintenance operation. In addition, the length is the number of bytes you wish to perform the cache maintenance operation. The function will determine if the length meets or exceeds 1/2 the cache size (as the cache contains 2 WAYs, half of the cache is in WAY0 and the other half in WAY1) and if so, will perform a cache maintenance "all" operation which is faster than performing the cache maintenance on a line-basis.

Cache Demotion: Cache region demotion - Demoting the cache mode reduces the cache function applied to a memory region from write-back to write-through to non-cacheable. The cache region demote function checks to see if the requested cache mode is higher than or equal to the current cache mode, and if so, will return an error. After a region is demoted, its cache mode can only be raised by a reset, which returns it to its default state. NOTE: The address/module assignment of the 16 subregions is device-specific and are detailed in the Chip Configuration chapter of the Kinetis reference manual. Some of the regions may not be used (non-cacheable), and some regions may not be capable of write-back.

To demote a cache region, simple call this function:

```
LMEM_DRV_CodeCacheDemoteRegion(uint32_t instance,
 lmem_cache_region_t region,
 lmem_cache_mode_t cacheMode);
```

The parameter region is of type lmem\_cache\_region\_t. This provides typedef enums for each of the 16 regions, starting with "kCacheRegion0" and ending with "kCacheRegion15". The parameter cacheMode is of type lmem\_cache\_mode\_t. This provides typedef enums for each of the cache modes: "kCacheNon-Cacheable", "kCacheWriteThrough", and "kCacheWriteBack".

Cache Enable/Disable: The cache enable function enables the PC or PS bus cache as well as the write buffer. However, before enabling these, the function first performs an invalidate all. The user should call this function if they wish to enable a particular bus cache. For example, to enable the Processor Code bus cache, call the function:

```
LMEM_DRV_CodeCacheEnable(uint32_t instance);
```

To enable the Processor System bus cache, call the function:

```
LMEM_DRV_SystemCacheEnable(uint32_t instance);
```

The cache disable function disables the PC or PS bus cache as well as the write buffer. Before disabling these, the function first performs a push all. The user should call this function if they wish to disable a particular bus cache. For example, to disable the Processor Code bus cache, call the function:

## LMEM Cache Driver

```
LMEM_DRV_CodeCacheDisable(uint32_t instance);
```

To Disable the Processor System bus cache, call the function:

```
LMEM_DRV_SystemCacheDisable(uint32_t instance);
```

## Variables

- LMEM\_Type \*const [g\\_lmemBase](#) [LMEM\_INSTANCE\_COUNT]  
*Table of base addresses for the LMEM instances.*

## Processor Code Bus Cache Control Peripheral Driver

- void [LMEM\\_DRV\\_CodeCacheInvalidateAll](#) (uint32\_t instance)  
*Invalidates the processor code bus cache.*
- void [LMEM\\_DRV\\_CodeCachePushAll](#) (uint32\_t instance)  
*Pushes all modified lines in the processor code bus cache.*
- void [LMEM\\_DRV\\_CodeCacheClearAll](#) (uint32\_t instance)  
*Clears the processor code bus cache.*
- void [LMEM\\_DRV\\_CodeCacheEnable](#) (uint32\_t instance)  
*Enables the processor code bus cache.*
- void [LMEM\\_DRV\\_CodeCacheDisable](#) (uint32\_t instance)  
*Disables the processor code bus cache.*
- void [LMEM\\_DRV\\_CodeCacheInvalidateLine](#) (uint32\_t instance, uint32\_t addr)  
*Invalidates a specific line in the processor code bus cache.*
- void [LMEM\\_DRV\\_CodeCacheInvalidateMultiLines](#) (uint32\_t instance, uint32\_t addr, uint32\_t length)  
*Invalidates multiple lines in the processor code bus cache.*
- void [LMEM\\_DRV\\_CodeCachePushLine](#) (uint32\_t instance, uint32\_t addr)  
*Pushes a specific modified line in the processor code bus cache.*
- void [LMEM\\_DRV\\_CodeCachePushMultiLines](#) (uint32\_t instance, uint32\_t addr, uint32\_t length)  
*Pushes multiple modified lines in the processor code bus cache.*
- void [LMEM\\_DRV\\_CodeCacheClearLine](#) (uint32\_t instance, uint32\_t addr)  
*Clears a specific line in the processor code bus cache.*
- void [LMEM\\_DRV\\_CodeCacheClearMultiLines](#) (uint32\_t instance, uint32\_t addr, uint32\_t length)  
*Clears multiple lines in the processor code bus cache.*
- [lmem\\_cache\\_status\\_t](#) [LMEM\\_DRV\\_CodeCacheDemoteRegion](#) (uint32\_t instance, [lmem\\_cache\\_region\\_t](#) region, [lmem\\_cache\\_mode\\_t](#) cacheMode)  
*Demotes the cache mode of a region in processor code bus cache.*

### 23.3.3 Function Documentation

#### 23.3.3.1 void LMEM\_DRV\_CodeCacheInvalidateAll ( uint32\_t instance )

This function invalidates the cache both ways, which means that it unconditionally clears valid bits and modifies bits of a cache entry.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the LMEM peripheral |
|-----------------|--------------------------------------------|

**23.3.3.2 void LMEM\_DRV\_CodeCachePushAll ( uint32\_t *instance* )**

This function pushes all modified lines in both ways (the entire cache). Pushes a cache entry if it is valid and modified, then clears the modify bit. If entry is not valid or not modified, leave as is. This action does not clear the valid bit. A cache push is synonymous with a cache flush.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the LMEM peripheral |
|-----------------|--------------------------------------------|

**23.3.3.3 void LMEM\_DRV\_CodeCacheClearAll ( uint32\_t *instance* )**

This function clears the entire cache and pushes (flushes) and invalidates the operation. Clear - Push a cache entry if it is valid and modified, then clear the valid and modify bits. If the entry is not valid or not modified, clear the valid bit.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the LMEM peripheral |
|-----------------|--------------------------------------------|

**23.3.3.4 void LMEM\_DRV\_CodeCacheEnable ( uint32\_t *instance* )**

This function enables the cache. The function first invalidates the entire cache, then enables both the cache and write buffer.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the LMEM peripheral |
|-----------------|--------------------------------------------|

**23.3.3.5 void LMEM\_DRV\_CodeCacheDisable ( uint32\_t *instance* )**

This function disables the cache and write buffer.

## LMEM Cache Driver

### Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the LMEM peripheral |
|-----------------|--------------------------------------------|

### 23.3.3.6 void LMEM\_DRV\_CodeCacheInvalidateLine ( uint32\_t *instance*, uint32\_t *addr* )

This function invalidates a specific line in the cache based on the physical address passed in by the user. Invalidate - Unconditionally clear valid and modify bits of a cache entry

### Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the LMEM peripheral |
| <i>addr</i>     | The physical address of the cache line     |

### 23.3.3.7 void LMEM\_DRV\_CodeCacheInvalidateMultiLines ( uint32\_t *instance*, uint32\_t *addr*, uint32\_t *length* )

This function invalidates multiple lines in the cache based on the physical address and length in bytes passed in by the user. If the function detects that the length meets or exceeds half the cache, then the function performs an entire cache invalidate function which is more efficient than invalidating the cache line-by-line. The need to check half the total amount of cache is due to the fact that the cache consists of two ways and that line commands based on the physical address searches both ways. Invalidate - Unconditionally clear valid and modify bits of a cache entry

### Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>instance</i> | The instance number of the LMEM peripheral             |
| <i>addr</i>     | The physical address of the cache line                 |
| <i>length</i>   | The length in bytes of the total amount of cache lines |

### 23.3.3.8 void LMEM\_DRV\_CodeCachePushLine ( uint32\_t *instance*, uint32\_t *addr* )

This function pushes a specific modified line based on the physical address passed in by the user. Push - Push a cache entry if it is valid and modified, then clear the modify bit. If entry not valid or not modified, leave as is. This action does not clear the valid bit. A cache push is synonymous with a cache flush.



## Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the LMEM peripheral |
| <i>addr</i>     | The physical address of the cache line     |

**23.3.3.9 void LMEM\_DRV\_CodeCachePushMultiLines ( uint32\_t *instance*, uint32\_t *addr*, uint32\_t *length* )**

This function pushes multiple modified lines in the cache based on the physical address and length in bytes passed in by the user. If the function detects that the length meets or exceeds half of the cache, the function performs an cache push function (which is more efficient than pushing the modified lines in the cache line-by-line). The need to check half the total amount of cache is due to the fact that the cache consists of two ways and that line commands based on the physical address searches both ways. Push - Push a cache entry if it is valid and modified, then clear the modify bit. If entry not valid or not modified, leave as is. This action does not clear the valid bit. A cache push is synonymous with a cache flush.

## Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>instance</i> | The instance number of the LMEM peripheral             |
| <i>addr</i>     | The physical address of the cache line                 |
| <i>length</i>   | The length in bytes of the total amount of cache lines |

**23.3.3.10 void LMEM\_DRV\_CodeCacheClearLine ( uint32\_t *instance*, uint32\_t *addr* )**

This function clears a specific line based on the physical address passed in by the user. Clear - Push a cache entry if it is valid and modified, then clear the valid and modify bits. If entry not valid or not modified, clear the valid bit.

## Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the LMEM peripheral |
| <i>addr</i>     | The physical address of the cache line     |

**23.3.3.11 void LMEM\_DRV\_CodeCacheClearMultiLines ( uint32\_t *instance*, uint32\_t *addr*, uint32\_t *length* )**

This function clears multiple lines in the cache based on the physical address and length in bytes passed in by the user. If the function detects that the length meets or exceeds half the total amount of cache, the function performs a cache clear function which is more efficient than clearing the lines in the cache line-by-line. The need to check half the total amount of cache is due to the fact that the cache consists of two ways and that line commands based on the physical address searches both ways. Clear - Push a cache

## LMEM Cache Driver

entry if it is valid and modified, then clear the valid and modify bits. If entry not valid or not modified, clear the valid bit.

Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>instance</i> | The instance number of the LMEM peripheral             |
| <i>addr</i>     | The physical address of the cache line                 |
| <i>length</i>   | The length in bytes of the total amount of cache lines |

### 23.3.3.12 `lmem_cache_status_t` LMEM\_DRV\_CodeCacheDemoteRegion ( `uint32_t instance`, `lmem_cache_region_t region`, `lmem_cache_mode_t cacheMode` )

This function allows the user to demote the cache mode of a region within the device's memory map. Demoting the cache mode reduces the cache function applied to a memory region from write-back to write-through to non-cacheable. The function checks to see if the requested cache mode is higher than or equal to the current cache mode, and if so, returns an error. After a region is demoted, its cache mode can only be raised by a reset, which returns it to its default state. To maintain cache coherency, changes to the cache mode should be completed while the address space being changed is not being accessed or the cache is disabled. Before a cache mode change, this function completes a cache clear all command to push and invalidate any cache entries that may have changed.

Parameters

|                  |                                                                       |
|------------------|-----------------------------------------------------------------------|
| <i>instance</i>  | The instance number of the LMEM peripheral                            |
| <i>region</i>    | The desired region to demote of type <code>lmem_cache_region_t</code> |
| <i>cacheMode</i> | The new, demoted cache mode of type <code>lmem_cache_mode_t</code>    |

Returns

`kStatus_LMEM_CACHE_Success` The cache clear operation was successful, or `kStatus_LMEM_CACHE_Busy` The cache is busy performing another operation `kStatus_LMEM_CACHE_Error` The requested cache mode is higher than or equal to the current cache mode

## 23.3.4 Variable Documentation

### 23.3.4.1 `LMEM_Type*` `const g_lmemBase[LMEM_INSTANCE_COUNT]`

---

## Chapter 24

# Universal Asynchronous Receiver/Transmitter (LPSCI)

### 24.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Universal Asynchronous Receiver/-Transmitter (LPSCI) block of Kinetis devices.

#### Modules

- [LPSCI HAL driver](#)
- [LPSCI Peripheral driver](#)

## LPSCI HAL driver

### 24.2 LPSCI HAL driver

#### 24.2.1 Overview

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

#### Files

- file [fsl\\_lpsci\\_hal.h](#)

#### Enumerations

- enum [lpsci\\_status\\_t](#)  
*Error codes for the LPSCI driver.*
- enum [lpsci\\_stop\\_bit\\_count\\_t](#) {  
[kLpsciOneStopBit](#) = 0U,  
[kLpsciTwoStopBit](#) = 1U }  
*LPSCI number of stop bits.*
- enum [lpsci\\_parity\\_mode\\_t](#) {  
[kLpsciParityDisabled](#) = 0x0U,  
[kLpsciParityEven](#) = 0x2U,  
[kLpsciParityOdd](#) = 0x3U }  
*LPSCI parity mode.*
- enum [lpsci\\_bit\\_count\\_per\\_char\\_t](#) {  
[kLpsci8BitsPerChar](#) = 0U,  
[kLpsci9BitsPerChar](#) = 1U }  
*LPSCI number of bits in a character.*
- enum [lpsci\\_operation\\_config\\_t](#) {  
[kLpsciOperates](#) = 0U,  
[kLpsciStops](#) = 1U }  
*LPSCI operation configuration constants.*
- enum [lpsci\\_receiver\\_source\\_t](#) {  
[kLpsciLoopBack](#) = 0U,  
[kLpsciSingleWire](#) = 1U }  
*LPSCI receiver source select mode.*
- enum [lpsci\\_wakeup\\_method\\_t](#) {  
[kLpsciIdleLineWake](#) = 0U,  
[kLpsciAddrMarkWake](#) = 1U }  
*LPSCI wakeup from standby method constants.*
- enum [lpsci\\_idle\\_line\\_select\\_t](#) {  
[kLpsciIdleLineAfterStartBit](#) = 0U,  
[kLpsciIdleLineAfterStopBit](#) = 1U }  
*LPSCI idle-line detect selection types.*
- enum [lpsci\\_break\\_char\\_length\\_t](#) {  
[kLpsciBreakChar10BitMinimum](#) = 0U,  
[kLpsciBreakChar13BitMinimum](#) = 1U }

- LPSCI break character length settings for transmit/detect.*

  - enum `lpsci_singlewire_txdir_t` {  
`kLpsciSinglewireTxdirIn` = 0U,  
`kLpsciSinglewireTxdirOut` = 1U }

*LPSCI single-wire mode transmit direction.*
- enum `lpsci_ir_tx_pulsewidth_t` {  
`kLpsciIrThreeSixteenthsWidth` = 0U,  
`kLpsciIrOneSixteenthWidth` = 1U,  
`kLpsciIrOneThirtysecondsWidth` = 2U,  
`kLpsciIrOneFourthWidth` = 3U }

*LPSCI infrared transmitter pulse width options.*
- enum `lpsci_status_flag_t` {  
`kLpsciTxDataRegEmpty` = 0U << LPSCI\_SHIFT | UART0\_S1\_TDRE\_SHIFT,  
`kLpsciTxComplete` = 0U << LPSCI\_SHIFT | UART0\_S1\_TC\_SHIFT,  
`kLpsciRxDataRegFull` = 0U << LPSCI\_SHIFT | UART0\_S1\_RDRF\_SHIFT,  
`kLpsciIdleLineDetect` = 0U << LPSCI\_SHIFT | UART0\_S1\_IDLE\_SHIFT,  
`kLpsciRxOverrun` = 0U << LPSCI\_SHIFT | UART0\_S1\_OR\_SHIFT,  
`kLpsciNoiseDetect` = 0U << LPSCI\_SHIFT | UART0\_S1\_NF\_SHIFT,  
`kLpsciFrameErr` = 0U << LPSCI\_SHIFT | UART0\_S1\_FE\_SHIFT,  
`kLpsciParityErr` = 0U << LPSCI\_SHIFT | UART0\_S1\_PF\_SHIFT,  
`kLpsciLineBreakDetect` = 1U << LPSCI\_SHIFT | UART0\_S2\_LBKDIF\_SHIFT,  
`kLpsciRxActiveEdgeDetect` = 1U << LPSCI\_SHIFT | UART0\_S2\_RXEDGIF\_SHIFT,  
`kLpsciRxActive` = 1U << LPSCI\_SHIFT | UART0\_S2\_RAF\_SHIFT }

*LPSCI status flags.*
- enum `lpsci_interrupt_t` {  
`kLpsciIntLinBreakDetect` = 0U << LPSCI\_SHIFT | UART0\_BDH\_LBKDIE\_SHIFT,  
`kLpsciIntRxActiveEdge` = 0U << LPSCI\_SHIFT | UART0\_BDH\_RXEDGIE\_SHIFT,  
`kLpsciIntTxDataRegEmpty` = 1U << LPSCI\_SHIFT | UART0\_C2\_TIE\_SHIFT,  
`kLpsciIntTxComplete` = 1U << LPSCI\_SHIFT | UART0\_C2\_TCIE\_SHIFT,  
`kLpsciIntRxDataRegFull` = 1U << LPSCI\_SHIFT | UART0\_C2\_RIE\_SHIFT,  
`kLpsciIntIdleLine` = 1U << LPSCI\_SHIFT | UART0\_C2\_ILIE\_SHIFT,  
`kLpsciIntRxOverrun` = 2U << LPSCI\_SHIFT | UART0\_C3\_ORIE\_SHIFT,  
`kLpsciIntNoiseErrFlag` = 2U << LPSCI\_SHIFT | UART0\_C3\_NEIE\_SHIFT,  
`kLpsciIntFrameErrFlag` = 2U << LPSCI\_SHIFT | UART0\_C3\_FEIE\_SHIFT,  
`kLpsciIntParityErrFlag` = 2U << LPSCI\_SHIFT | UART0\_C3\_PEIE\_SHIFT }

*LPSCI interrupt configuration structure, default settings are 0 (disabled).*

## LPSCI Common Configurations

- void `LPSCI_HAL_Init` (UART0\_Type \*base)  
*Initializes the LPSCI controller.*
- static void `LPSCI_HAL_EnableTransmitter` (UART0\_Type \*base)  
*Enables the LPSCI transmitter.*
- static void `LPSCI_HAL_DisableTransmitter` (UART0\_Type \*base)  
*Disables the LPSCI transmitter.*
- static bool `LPSCI_HAL_IsTransmitterEnabled` (UART0\_Type \*base)

## LPSCI HAL driver

- Gets the LPSCI transmitter enabled/disabled configuration setting.*

  - static void [LPSCI\\_HAL\\_EnableReceiver](#) (UART0\_Type \*base)

*Enables the LPSCI receiver.*
- static void [LPSCI\\_HAL\\_DisableReceiver](#) (UART0\_Type \*base)

*Disables the LPSCI receiver.*
- static bool [LPSCI\\_HAL\\_IsReceiverEnabled](#) (UART0\_Type \*base)

*Gets the LPSCI receiver enabled/disabled configuration setting.*
- [lpsci\\_status\\_t](#) [LPSCI\\_HAL\\_SetBaudRate](#) (UART0\_Type \*base, uint32\_t sourceClockInHz, uint32\_t baudRate)

*Configures the LPSCI baud rate.*
- void [LPSCI\\_HAL\\_SetBaudRateDivisor](#) (UART0\_Type \*base, uint16\_t baudRateDivisor)

*Sets the LPSCI baud rate modulo divisor value.*
- static void [LPSCI\\_HAL\\_SetBitCountPerChar](#) (UART0\_Type \*base, [lpsci\\_bit\\_count\\_per\\_char\\_t](#) bitCountPerChar)

*Configures the number of bits per character in the LPSCI controller.*
- void [LPSCI\\_HAL\\_SetParityMode](#) (UART0\_Type \*base, [lpsci\\_parity\\_mode\\_t](#) parityMode)

*Configures the parity mode in LPSCI controller.*

## LPSCI Interrupts and DMA

- void [LPSCI\\_HAL\\_SetIntMode](#) (UART0\_Type \*base, [lpsci\\_interrupt\\_t](#) interrupt, bool enable)

*Configures the LPSCI module interrupts to enable/disable various interrupt sources.*
- bool [LPSCI\\_HAL\\_GetIntMode](#) (UART0\_Type \*base, [lpsci\\_interrupt\\_t](#) interrupt)

*Returns whether the LPSCI module interrupts is enabled/disabled.*
- static uint32\_t [LPSCI\\_HAL\\_GetDataRegAddr](#) (UART0\_Type \*base)

*Get LPSCI tx/rx data register address.*

## LPSCI Transfer Functions

- static void [LPSCI\\_HAL\\_Putchar](#) (UART0\_Type \*base, uint8\_t data)

*This function allows the user to send an 8-bit character from the LPSCI data register.*
- void [LPSCI\\_HAL\\_Putchar9](#) (UART0\_Type \*base, uint16\_t data)

*This function allows the user to send a 9-bit character from the LPSCI data register.*
- void [LPSCI\\_HAL\\_Putchar10](#) (UART0\_Type \*base, uint16\_t data)

*This function allows the user to send a 10-bit character from the LPSCI data register.*
- static void [LPSCI\\_HAL\\_Getchar](#) (UART0\_Type \*base, uint8\_t \*readData)

*This function gets a received 8-bit character from the LPSCI data register.*
- void [LPSCI\\_HAL\\_Getchar9](#) (UART0\_Type \*base, uint16\_t \*readData)

*This function gets a received 9-bit character from the LPSCI data register.*
- void [LPSCI\\_HAL\\_Getchar10](#) (UART0\_Type \*base, uint16\_t \*readData)

*This function gets a received 10-bit character from the LPSCI data register.*
- void [LPSCI\\_HAL\\_SendDataPolling](#) (UART0\_Type \*base, const uint8\_t \*txBuff, uint32\_t txSize)

*Send out multiple bytes of data using polling method.*
- [lpsci\\_status\\_t](#) [LPSCI\\_HAL\\_ReceiveDataPolling](#) (UART0\_Type \*base, uint8\_t \*rxBuff, uint32\_t rxSize)

*Receive multiple bytes of data using polling method.*

## LPSCI Status Flags

- bool [LPSCI\\_HAL\\_GetStatusFlag](#) (UART0\_Type \*base, [lpsci\\_status\\_flag\\_t](#) statusFlag)  
*Gets all LPSCI status flag states.*
- [lpsci\\_status\\_t LPSCI\\_HAL\\_ClearStatusFlag](#) (UART0\_Type \*base, [lpsci\\_status\\_flag\\_t](#) statusFlag)  
*Clears an individual and specific LPSCI status flag.*

## LPSCI Special Feature Configurations

- static void [LPSCI\\_HAL\\_SetWaitModeOperation](#) (UART0\_Type \*base, [lpsci\\_operation\\_config\\_t](#) mode)  
*Configures the LPSCI to either operate or cease to operate in WAIT mode.*
- static [lpsci\\_operation\\_config\\_t LPSCI\\_HAL\\_GetWaitModeOperation](#) (UART0\_Type \*base)  
*Determines if the LPSCI operates or ceases to operate in WAIT mode.*
- static void [LPSCI\\_HAL\\_SetLoopCmd](#) (UART0\_Type \*base, bool enable)  
*Configures the LPSCI loopback operation.*
- static void [LPSCI\\_HAL\\_SetReceiverSource](#) (UART0\_Type \*base, [lpsci\\_receiver\\_source\\_t](#) source)  
*Configures the LPSCI single-wire operation.*
- static void [LPSCI\\_HAL\\_SetTransmitterDir](#) (UART0\_Type \*base, [lpsci\\_singlewire\\_txdir\\_t](#) direction)  
*Configures the LPSCI transmit direction while in single-wire mode.*
- [lpsci\\_status\\_t LPSCI\\_HAL\\_PutReceiverInStandbyMode](#) (UART0\_Type \*base)  
*Places the LPSCI receiver in standby mode.*
- static void [LPSCI\\_HAL\\_PutReceiverInNormalMode](#) (UART0\_Type \*base)  
*Places the LPSCI receiver in normal mode (disable standby mode operation).*
- static bool [LPSCI\\_HAL\\_IsReceiverInStandby](#) (UART0\_Type \*base)  
*Determines if the LPSCI receiver is currently in standby mode.*
- static void [LPSCI\\_HAL\\_SetReceiverWakeupMethod](#) (UART0\_Type \*base, [lpsci\\_wakeup\\_method\\_t](#) method)  
*Selects the LPSCI receiver wakeup method (idle-line or address-mark) from standby mode.*
- static [lpsci\\_wakeup\\_method\\_t LPSCI\\_HAL\\_GetReceiverWakeupMethod](#) (UART0\_Type \*base)  
*Gets the LPSCI receiver wakeup method (idle-line or address-mark) from standby mode.*
- void [LPSCI\\_HAL\\_ConfigIdleLineDetect](#) (UART0\_Type \*base, uint8\_t idleLine, uint8\_t rxWakeIdleDetect)  
*Configures the operation options of the LPSCI idle line detect.*
- static void [LPSCI\\_HAL\\_SetBreakCharTransmitLength](#) (UART0\_Type \*base, [lpsci\\_break\\_char\\_length\\_t](#) length)  
*Configures the LPSCI break character transmit length.*
- static void [LPSCI\\_HAL\\_SetBreakCharDetectLength](#) (UART0\_Type \*base, [lpsci\\_break\\_char\\_length\\_t](#) length)  
*Configures the LPSCI break character detect length.*
- static void [LPSCI\\_HAL\\_SetBreakCharCmd](#) (UART0\_Type \*base, bool enable)  
*Configures the LPSCI transmit send break character operation.*
- void [LPSCI\\_HAL\\_SetMatchAddress](#) (UART0\_Type \*base, bool matchAddrMode1, bool matchAddrMode2, uint8\_t matchAddrValue1, uint8\_t matchAddrValue2)  
*Configures the LPSCI match address mode control operation.*

## LPSCI HAL driver

### 24.2.2 Enumeration Type Documentation

#### 24.2.2.1 enum lpsci\_status\_t

#### 24.2.2.2 enum lpsci\_stop\_bit\_count\_t

These constants define the number of allowable stop bits to configure in a LPSCI base.

Enumerator

*kLpsciOneStopBit* one stop bit  
*kLpsciTwoStopBit* two stop bits

#### 24.2.2.3 enum lpsci\_parity\_mode\_t

These constants define the LPSCI parity mode options: disabled or enabled of type even or odd.

Enumerator

*kLpsciParityDisabled* parity disabled  
*kLpsciParityEven* parity enabled, type even, bit setting: PE|PT = 10  
*kLpsciParityOdd* parity enabled, type odd, bit setting: PE|PT = 11

#### 24.2.2.4 enum lpsci\_bit\_count\_per\_char\_t

These constants define the number of allowable data bits per LPSCI character. Note, check the LPSCI documentation to determine if the desired LPSCI base supports the desired number of data bits per LPSCI character.

Enumerator

*kLpsci8BitsPerChar* 8-bit data characters  
*kLpsci9BitsPerChar* 9-bit data characters

#### 24.2.2.5 enum lpsci\_operation\_config\_t

This provides constants for LPSCI operational states: "operates normally" or "stops/ceases operation"

Enumerator

*kLpsciOperates* LPSCI continues to operate normally.  
*kLpsciStops* LPSCI ceases operation.



### 24.2.2.6 enum lpsci\_receiver\_source\_t

Enumerator

*kLpsciLoopBack* Internal loop back mode.

*kLpsciSingleWire* Single wire mode.

### 24.2.2.7 enum lpsci\_wakeup\_method\_t

This provides constants for the two LPSCI wakeup methods: idle-line or address-mark.

Enumerator

*kLpsciIdleLineWake* The idle-line wakes LPSCI receiver from standby.

*kLpsciAddrMarkWake* The address-mark wakes LPSCI receiver from standby.

### 24.2.2.8 enum lpsci\_idle\_line\_select\_t

This provides constants for the LPSCI idle character bit-count start: either after start or stop bit.

Enumerator

*kLpsciIdleLineAfterStartBit* LPSCI idle character bit count start after start bit.

*kLpsciIdleLineAfterStopBit* LPSCI idle character bit count start after stop bit.

### 24.2.2.9 enum lpsci\_break\_char\_length\_t

This provides constants for the LPSCI break character length for both transmission and detection purposes. Note that the actual maximum bit times may vary depending on the LPSCI base.

Enumerator

*kLpsciBreakChar10BitMinimum* LPSCI 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)

*kLpsciBreakChar13BitMinimum* LPSCI 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)

### 24.2.2.10 enum lpsci\_singlewire\_txdir\_t

This provides constants for the LPSCI transmit direction when configured for single-wire mode. The transmit line TXDIR is either an input or output.

## LPSCI HAL driver

Enumerator

*kLpisciSinglewireTxdirIn* LPSCI Single-Wire mode TXDIR input.  
*kLpisciSinglewireTxdirOut* LPSCI Single-Wire mode TXDIR output.

### 24.2.2.11 enum lpisci\_ir\_tx\_pulsewidth\_t

This provides constants for the LPSCI infrared (IR) pulse widths. Options include 3/16, 1/16, 1/32, and 1/4 pulse widths.

Enumerator

*kLpisciIrThreeSixteenthsWidth* 3/16 pulse  
*kLpisciIrOneSixteenthWidth* 1/16 pulse  
*kLpisciIrOneThirtysecondsWidth* 1/32 pulse  
*kLpisciIrOneFourthWidth* 1/4 pulse

### 24.2.2.12 enum lpisci\_status\_flag\_t

This provides constants for the LPSCI status flags for use in the LPSCI functions.

Enumerator

*kLpisciTxDataRegEmpty* Tx data register empty flag, sets when Tx buffer is empty.  
*kLpisciTxComplete* Transmission complete flag, sets when transmission activity complete.  
*kLpisciRxDataRegFull* Rx data register full flag, sets when the receive data buffer is full.  
*kLpisciIdleLineDetect* Idle line detect flag, sets when idle line detected.  
*kLpisciRxOverrun* Rxr Overrun, sets when new data is received before data is read from receive register.  
*kLpisciNoiseDetect* Rxr takes 3 samples of each received bit. If any of these samples differ, noise flag sets  
*kLpisciFrameErr* Frame error flag, sets if logic 0 was detected where stop bit expected.  
*kLpisciParityErr* If parity enabled, sets upon parity error detection.  
*kLpisciLineBreakDetect* LIN break detect interrupt flag, sets when LIN break char detected and LIN circuit enabled.  
*kLpisciRxActiveEdgeDetect* Rx pin active edge interrupt flag, sets when active edge detected.  
*kLpisciRxActive* Receiver Active Flag (RAF), sets at beginning of valid start bit.

### 24.2.2.13 enum lpisci\_interrupt\_t

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

Enumerator

*kLpisciIntLinBreakDetect* LIN break detect.

- kLpSciIntRxActiveEdge* RX Active Edge.
- kLpSciIntTxDataRegEmpty* Transmit data register empty.
- kLpSciIntTxComplete* Transmission complete.
- kLpSciIntRxDataRegFull* Receiver data register full.
- kLpSciIntIdleLine* Idle line.
- kLpSciIntRxOverrun* Receiver Overrun.
- kLpSciIntNoiseErrFlag* Noise error flag.
- kLpSciIntFrameErrFlag* Framing error flag.
- kLpSciIntParityErrFlag* Parity error flag.

### 24.2.3 Function Documentation

#### 24.2.3.1 void LPSCI\_HAL\_Init ( UART0\_Type \* base )

This function initializes the module to a known state.

Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | LPSCI module base pointer. |
|-------------|----------------------------|

#### 24.2.3.2 static void LPSCI\_HAL\_EnableTransmitter ( UART0\_Type \* base ) [inline], [static]

This function allows the user to enable the LPSCI transmitter.

Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | LPSCI module base pointer. |
|-------------|----------------------------|

#### 24.2.3.3 static void LPSCI\_HAL\_DisableTransmitter ( UART0\_Type \* base ) [inline], [static]

This function allows the user to disable the LPSCI transmitter.

Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | LPSCI module base pointer. |
|-------------|----------------------------|

#### 24.2.3.4 static bool LPSCI\_HAL\_IsTransmitterEnabled ( UART0\_Type \* base ) [inline], [static]

This function allows the user to get the setting of the LPSCI transmitter.

## LPSCI HAL driver

### Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | LPSCI module base pointer. |
|-------------|----------------------------|

### Returns

The state of LPSCI transmitter enable(true)/disable(false) setting.

**24.2.3.5 static void LPSCI\_HAL\_EnableReceiver ( UART0\_Type \* *base* ) [inline], [static]**

This function allows the user to enable the LPSCI receiver.

### Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | LPSCI module base pointer. |
|-------------|----------------------------|

**24.2.3.6 static void LPSCI\_HAL\_DisableReceiver ( UART0\_Type \* *base* ) [inline], [static]**

This function allows the user to disable the LPSCI receiver.

### Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | LPSCI module base pointer. |
|-------------|----------------------------|

**24.2.3.7 static bool LPSCI\_HAL\_IsReceiverEnabled ( UART0\_Type \* *base* ) [inline], [static]**

This function allows the user to get the setting of the LPSCI receiver.

### Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | LPSCI module base pointer. |
|-------------|----------------------------|

### Returns

The state of LPSCI receiver enable(true)/disable(false) setting.

**24.2.3.8** `lpsci_status_t LPSCI_HAL_SetBaudRate ( UART0_Type * base, uint32_t sourceClockInHz, uint32_t baudRate )`

This function programs the LPSCI 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 LPSCI bases it is required that the transmitter/receiver be disabled before calling this function. Generally this is applied to all LPSCIs to ensure safe operation.

Parameters

|                        |                                 |
|------------------------|---------------------------------|
| <i>base</i>            | LPSCI module base pointer.      |
| <i>sourceClockInHz</i> | LPSCI source input clock in Hz. |
| <i>baudRate</i>        | LPSCI desired baud rate.        |

Returns

An error code or `kStatus_LPSCI_Success`

**24.2.3.9** `void LPSCI_HAL_SetBaudRateDivisor ( UART0_Type * base, 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 [LPSCI\\_HAL\\_SetBaudRate\(\)](#) function, as the divisor is already known.

Parameters

|                        |                                            |
|------------------------|--------------------------------------------|
| <i>base</i>            | LPSCI module base pointer.                 |
| <i>baudRateDivisor</i> | The baud rate modulo division "SBR" value. |

**24.2.3.10** `static void LPSCI_HAL_SetBitCountPerChar ( UART0_Type * base, lpsci_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 `lpsci_bit_count_per_char_t`.

## LPSCI HAL driver

### Parameters

|                        |                                                                     |
|------------------------|---------------------------------------------------------------------|
| <i>base</i>            | LPSCI module base pointer.                                          |
| <i>bitCountPerChar</i> | Number of bits per char (8, 9, or 10, depending on the LPSCI base). |

### 24.2.3.11 void LPSCI\_HAL\_SetParityMode ( UART0\_Type \* *base*, lpsci\_parity\_mode\_t *parityMode* )

This function allows the user to configure the parity mode of the LPSCI controller to disable it or enable it for even parity or for odd parity.

### Parameters

|                   |                                                                               |
|-------------------|-------------------------------------------------------------------------------|
| <i>base</i>       | LPSCI module base pointer.                                                    |
| <i>parityMode</i> | Parity mode setting (enabled, disable, odd, even - see parity_mode_t struct). |

### 24.2.3.12 void LPSCI\_HAL\_SetIntMode ( UART0\_Type \* *base*, lpsci\_interrupt\_t *interrupt*, bool *enable* )

### Parameters

|                  |                                     |
|------------------|-------------------------------------|
| <i>base</i>      | LPSCI module base pointer.          |
| <i>interrupt</i> | LPSCI interrupt configuration data. |
| <i>enable</i>    | true: enable, false: disable.       |

### 24.2.3.13 bool LPSCI\_HAL\_GetIntMode ( UART0\_Type \* *base*, lpsci\_interrupt\_t *interrupt* )

### Parameters

|                  |                                     |
|------------------|-------------------------------------|
| <i>base</i>      | LPSCI module base pointer.          |
| <i>interrupt</i> | LPSCI interrupt configuration data. |

### Returns

true: enable, false: disable.

```
24.2.3.14 static uint32_t LPSCI_HAL_GetDataRegAddr (UART0_Type * base)
 [inline], [static]
```

This function is used for DMA transfer.

## LPSCI HAL driver

### Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | LPSCI module base address. |
|-------------|----------------------------|

### Returns

LPSCI tx/rx data register address.

**24.2.3.15 static void LPSCI\_HAL\_Putchar ( UART0\_Type \* *base*, uint8\_t *data* )  
[inline], [static]**

### Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | LPSCI module base pointer.      |
| <i>data</i> | The data to send of size 8-bit. |

**24.2.3.16 void LPSCI\_HAL\_Putchar9 ( UART0\_Type \* *base*, uint16\_t *data* )**

### Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | LPSCI module base pointer.      |
| <i>data</i> | The data to send of size 9-bit. |

**24.2.3.17 void LPSCI\_HAL\_Putchar10 ( UART0\_Type \* *base*, uint16\_t *data* )**

### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | LPSCI module base pointer.       |
| <i>data</i> | The data to send of size 10-bit. |

**24.2.3.18 static void LPSCI\_HAL\_Getchar ( UART0\_Type \* *base*, uint8\_t \* *readData* )  
[inline], [static]**



Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>base</i>     | LPSCI module base pointer.                               |
| <i>readData</i> | The received data read from data register of size 8-bit. |

**24.2.3.19 void LPSCI\_HAL\_Getchar9 ( UART0\_Type \* *base*, uint16\_t \* *readData* )**

Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>base</i>     | LPSCI module base pointer.                               |
| <i>readData</i> | The received data read from data register of size 9-bit. |

**24.2.3.20 void LPSCI\_HAL\_Getchar10 ( UART0\_Type \* *base*, uint16\_t \* *readData* )**

Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>base</i>     | LPSCI module base pointer.                                |
| <i>readData</i> | The received data read from data register of size 10-bit. |

**24.2.3.21 void LPSCI\_HAL\_SendDataPolling ( UART0\_Type \* *base*, const uint8\_t \* *txBuff*, uint32\_t *txSize* )**

This function only supports 8-bit transaction.

Parameters

|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>base</i>   | LPSCI module base pointer.                          |
| <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.            |

**24.2.3.22 lpsci\_status\_t LPSCI\_HAL\_ReceiveDataPolling ( UART0\_Type \* *base*, uint8\_t \* *rxBuff*, uint32\_t *rxSize* )**

This function only supports 8-bit transaction.

## LPSCI HAL driver

### Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | LPSCI module base pointer.                              |
| <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.

#### 24.2.3.23 **bool LPSCI\_HAL\_GetStatusFlag ( UART0\_Type \* *base*, lpsci\_status\_flag\_t *statusFlag* )**

### Parameters

|                   |                            |
|-------------------|----------------------------|
| <i>base</i>       | LPSCI module base pointer. |
| <i>statusFlag</i> | Status flag name.          |

#### 24.2.3.24 **lpsci\_status\_t LPSCI\_HAL\_ClearStatusFlag ( UART0\_Type \* *base*, lpsci\_status\_flag\_t *statusFlag* )**

This function allows the user to clear an individual and specific LPSCI status flag. Refer to structure definition lpsci\_status\_flag\_t for list of status bits.

### Parameters

|                   |                                         |
|-------------------|-----------------------------------------|
| <i>base</i>       | LPSCI module base pointer.              |
| <i>statusFlag</i> | The desired LPSCI status flag to clear. |

### Returns

An error code or kStatus\_LPSCI\_Success.

#### 24.2.3.25 **static void LPSCI\_HAL\_SetWaitModeOperation ( UART0\_Type \* *base*, lpsci\_operation\_config\_t *mode* ) [inline], [static]**

The function configures the LPSCI to either operate or cease to operate when WAIT mode is entered.

Parameters

|             |                                                                             |
|-------------|-----------------------------------------------------------------------------|
| <i>base</i> | LPSCI module base pointer.                                                  |
| <i>mode</i> | The LPSCI WAIT mode operation - operates or ceases to operate in WAIT mode. |

**24.2.3.26 static lpsci\_operation\_config\_t LPSCI\_HAL\_GetWaitModeOperation ( UART0\_Type \* *base* ) [inline], [static]**

This function returns kLpSciOperates if the LPSCI has been configured to operate in WAIT mode. Else it returns KLpSciStops if the LPSCI has been configured to cease-to-operate in WAIT mode.

Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | LPSCI module base pointer. |
|-------------|----------------------------|

Returns

The LPSCI WAIT mode operation configuration, returns either kLpSciOperates or KLpSciStops.

**24.2.3.27 static void LPSCI\_HAL\_SetLoopCmd ( UART0\_Type \* *base*, bool *enable* ) [inline], [static]**

This function enables or disables the LPSCI loopback operation.

Parameters

|               |                                                                                   |
|---------------|-----------------------------------------------------------------------------------|
| <i>base</i>   | LPSCI module base pointer.                                                        |
| <i>enable</i> | The LPSCI loopback mode configuration, either disabled (false) or enabled (true). |

**24.2.3.28 static void LPSCI\_HAL\_SetReceiverSource ( UART0\_Type \* *base*, lpsci\_receiver\_source\_t *source* ) [inline], [static]**

This function enables or disables the LPSCI single-wire operation. In some LPSCI bases it is required that the transmitter/receiver be disabled before calling this function. This may be applied to all LPSCIs to ensure safe operation.

## LPSCI HAL driver

Parameters

|               |                                                                                      |
|---------------|--------------------------------------------------------------------------------------|
| <i>base</i>   | LPSCI module base pointer.                                                           |
| <i>source</i> | The LPSCI single-wire mode configuration, either disabled (false) or enabled (true). |

### 24.2.3.29 static void LPSCI\_HAL\_SetTransmitterDir ( UART0\_Type \* *base*, lpsci\_singlewire\_txdir\_t *direction* ) [inline], [static]

This function configures the transmitter direction when the LPSCI is configured for single-wire operation.

Parameters

|                  |                                                                                                                                                            |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | LPSCI module base pointer.                                                                                                                                 |
| <i>direction</i> | The LPSCI single-wire mode transmit direction configuration of type lpsci_singlewire_txdir_t (either kLpSciSinglewireTxdirIn or kLpSciSinglewireTxdirOut). |

### 24.2.3.30 lpsci\_status\_t LPSCI\_HAL\_PutReceiverInStandbyMode ( UART0\_Type \* *base* )

This function, when called, places the LPSCI receiver into standby mode. In some LPSCI bases, there are conditions that must be met before placing Rx in standby mode. Before placing LPSCI 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 LPSCI 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>base</i> | LPSCI module base pointer. |
|-------------|----------------------------|

Returns

Error code or kStatus\_LPSCI\_Success.

### 24.2.3.31 static void LPSCI\_HAL\_PutReceiverInNormalMode ( UART0\_Type \* *base* ) [inline], [static]

This function, when called, places the LPSCI receiver into normal mode and out of standby mode.

Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | LPSCI module base pointer. |
|-------------|----------------------------|

**24.2.3.32 static bool LPSCI\_HAL\_IsReceiverInStandby ( UART0\_Type \* *base* )  
[inline], [static]**

This function determines the state of the LPSCI receiver. If it returns true, this means that the LPSCI receiver is in standby mode; if it returns false, the LPSCI receiver is in normal mode.

Parameters

|             |                            |
|-------------|----------------------------|
| <i>base</i> | LPSCI module base pointer. |
|-------------|----------------------------|

Returns

The LPSCI receiver is in normal mode (false) or standby mode (true).

**24.2.3.33 static void LPSCI\_HAL\_SetReceiverWakeupMethod ( UART0\_Type \* *base*,  
lpsci\_wakeup\_method\_t *method* ) [inline], [static]**

This function configures the wakeup method of the LPSCI receiver from standby mode. The options are idle-line wake or address-mark wake.

Parameters

|               |                                                                                                                          |
|---------------|--------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | LPSCI module base pointer.                                                                                               |
| <i>method</i> | The LPSCI receiver wakeup method options: kLpsciIdleLineWake - Idle-line wake or kLpsciAddrMarkWake - address-mark wake. |

**24.2.3.34 static lpsci\_wakeup\_method\_t LPSCI\_HAL\_GetReceiverWakeupMethod ( UART0\_Type \* *base* ) [inline], [static]**

This function returns how the LPSCI receiver is configured to wake from standby mode. The wake method options that can be returned are kLpsciIdleLineWake or kLpsciAddrMarkWake.

Parameters

---

## LPSCI HAL driver

|             |                            |
|-------------|----------------------------|
| <i>base</i> | LPSCI module base pointer. |
|-------------|----------------------------|

Returns

The LPSCI receiver wakeup from standby method, false: kLpsciIdleLineWake (idle-line wake) or true: kLpsciAddrMarkWake (address-mark wake).

### 24.2.3.35 void LPSCI\_HAL\_ConfigIdleLineDetect ( UART0\_Type \* *base*, uint8\_t *idleLine*, uint8\_t *rxWakeIdleDetect* )

This function allows the user to configure the LPSCI 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 `lpsci_idle_line_config_t`.

Parameters

|                          |                                                                                                                                                                                                                                                   |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | LPSCI module base pointer.                                                                                                                                                                                                                        |
| <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 |

### 24.2.3.36 static void LPSCI\_HAL\_SetBreakCharTransmitLength ( UART0\_Type \* *base*, lpsci\_break\_char\_length\_t *length* ) [inline], [static]

This function allows the user to configure the LPSCI break character transmit length. Refer to the typedef `lpsci_break_char_length_t` for setting options. In some LPSCI bases it is required that the transmitter be disabled before calling this function. This may be applied to all LPSCIs to ensure safe operation.

Parameters

|               |                                                                                                                                                    |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | LPSCI module base pointer.                                                                                                                         |
| <i>length</i> | The LPSCI break character length setting of type <code>lpsci_break_char_length_t</code> , either a minimum 10-bit times or a minimum 13-bit times. |

### 24.2.3.37 static void LPSCI\_HAL\_SetBreakCharDetectLength ( UART0\_Type \* *base*, lpsci\_break\_char\_length\_t *length* ) [inline], [static]

This function allows the user to configure the LPSCI break character detect length. Refer to the typedef `lpsci_break_char_length_t` for setting options.

Parameters

|               |                                                                                                                                                    |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | LPSCI module base pointer.                                                                                                                         |
| <i>length</i> | The LPSCI break character length setting of type <code>lpsci_break_char_length_t</code> , either a minimum 10-bit times or a minimum 13-bit times. |

**24.2.3.38 static void LPSCI\_HAL\_SetBreakCharCmd ( UART0\_Type \* *base*, bool *enable* )  
[inline], [static]**

This function allows the user to queue a LPSCI 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>base</i>   | LPSCI module base pointer.                                                                                                                                                                |
| <i>enable</i> | If false, the LPSCI normal/queue break character setting is disabled, which configures the LPSCI for normal transmitter operation. If true, a break character is queued for transmission. |

**24.2.3.39 void LPSCI\_HAL\_SetMatchAddress ( UART0\_Type \* *base*, bool *matchAddrMode1*, bool *matchAddrMode2*, uint8\_t *matchAddrValue1*, uint8\_t *matchAddrValue2* )**

(Note: Feature available on select LPSCI bases)

The function allows the user to configure the LPSCI 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>base</i>            | LPSCI module base pointer.                                                |
| <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. |

## LPSCI HAL driver

|                         |                                                              |
|-------------------------|--------------------------------------------------------------|
| <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. |



## 24.3 LPSCI Peripheral driver

### 24.3.1 Overview

The section describes the programming interface of the LPSCI Peripheral driver. The LPSCI peripheral driver transfers data to and from external devices on the Universal Asynchronous Receiver/Transmitter (LPSCI) serial bus with a single function call.

### 24.3.2 LPSCI Device structures

The driver uses an instantiation of the `lpsci_state_t` structure to maintain the current state of a particular LPSCI instance module driver. This structure holds data used by the LPSCI 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 LPSCI driver populates the structure members.

### 24.3.3 LPSCI User configuration structures

The LPSCI driver uses instances of the user configuration structure, `lpsci_user_config_t`, for the LPSCI driver. As a result, the most common settings of the LPSCI are configured with a single function call. Settings include: LPSCI baud rate; LPSCI parity mode: disabled (default), or even or odd; the number of stop bits; the number of bits per data word.

### 24.3.4 LPSCI Initialization

1. To initialize the LPSCI driver, call the `LPSCI_DRV_Init()` function and pass the instance number of the relevant LPSCI peripheral, memory for the run-time state structure, and a pointer to the user configuration structure. For example, to use LPSCI0 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 `lpsci_user_config_t` as shown here:

```
// LPSCI configuration structure
typedef struct LpSciUserConfig {
 uint32_t baudRate;
 lpsci_parity_mode_t parityMode;
 lpsci_stop_bit_count_t stopBitCount;
 lpsci_bit_count_per_char_t bitCountPerChar;
} lpsci_user_config_t;
```

Typically, the `lpsci_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 `lpsci_user_config_t` instantiation can be easily modified to configure the LPSCI Peripheral driver either to a different baud rate or character transfer features. This is an example code to set up a user LPSCI configuration instantiation:

## LPSCI Peripheral driver

```
lpsci_user_config_t lpsciConfig;
lpsciConfig.baudRate = 9600;
lpsciConfig.bitCountPerChar = kLpSci8BitsPerChar;
lpsciConfig.parityMode = kLpSciParityDisabled;
lpsciConfig.stopBitCount = kLpSciOneStopBit;
```

This example shows how to call the `LPSCI_DRV_Init()` function given the user configuration structure and the LPSCI instance 0.

```
uint32_t lpsciInstance = 0;
lpsci_state_t lpsciState; // user provides memory for the driver state structure

LPSCI_DRV_Init(lpsciInstance, &lpsciConfig, &lpsciState);
```

### 24.3.5 LPSCI 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 `LPSCI_DRV_SendData()` and `LPSCI_DRV_ReceiveData()` functions.

The blocking (async) transmit and receive functions include the `LPSCI_DRV_SendDataBlocking()` and `LPSCI_DRV_ReceiveDataBlocking()` functions.

In all cases mentioned here, the functions are interrupt-driven.

These code examples show how to use previously mentioned functions and assume that the LPSCI 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 LPSCI_DRV_ReceiveData function

uint32_t byteCount = sizeof(sourceBuff);
uint32_t rxRemainingSize = sizeof(readBuffer);

// for each use there, set timeout as "1"
// lpsciState is the run-time state. Pass in memory for this
// declared previously in the initialization chapter
LPSCI_DRV_SendDataBlocking(&lpsciState, sourceBuff, byteCount, 1); // function
 won't return until transmit is complete
LPSCI_DRV_ReceiveDataBlocking(&lpsciState, 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 txBytesRemaining, rxBytesRemaining;

// assume pTxBuff and txSize have been initialized
LPSCI_DRV_SendData(&lpsciState, pTxBuff, txSize);

// 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 (LPSCI_DRV_GetTransmitStatus(&lpsciState, &txBytesRemaining) ==
 kStatus_LPSCI_TxBusy);

// for receive, assume rxBuff is set up to receive data and rxSize is initialized
LPSCI_DRV_ReceiveData(&lpsciState, rxBuff, rxSize);

// 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 (LPSCI_DRV_GetReceiveStatus(&lpsciState, &rxBytesRemaining) ==
 kStatus_LPSCI_RxBusy);
```

### Files

- file [fsl\\_lpsci\\_driver.h](#)  
*This driver is for UART0 if UART0 is a separate chapter in the chip reference manual.*

### Data Structures

- struct [lpsci\\_dma\\_state\\_t](#)  
*Runtime state structure for LPSCI driver with DMA. [More...](#)*
- struct [lpsci\\_dma\\_user\\_config\\_t](#)  
*User configuration structure for the LPSCI driver. [More...](#)*
- struct [lpsci\\_state\\_t](#)  
*Runtime state of the LPSCI driver. [More...](#)*
- struct [lpsci\\_user\\_config\\_t](#)  
*User configuration structure for the LPSCI driver. [More...](#)*

### Typedefs

- typedef void(\* [lpsci\\_rx\\_callback\\_t](#))(uint32\_t instance, void \*lpsciState)  
*LPSCI receive callback function type.*
- typedef void(\* [lpsci\\_tx\\_callback\\_t](#))(uint32\_t instance, void \*lpsciState)  
*LPSCI transmit callback function type.*

### Variables

- UART0\_Type \*const [g\\_lpsciBase](#) [UART0\_INSTANCE\_COUNT]  
*Table of base addresses for LPSCI instances.*
- UART0\_Type \*const [g\\_lpsciBase](#) [UART0\_INSTANCE\_COUNT]  
*Table of base addresses for LPSCI instances.*
- const IRQn\_Type [g\\_lpsciRxTxIrqId](#) [UART0\_INSTANCE\_COUNT]  
*Table to save LPSCI IRQ enumeration numbers defined in CMSIS header file.*

## LPSCI Peripheral driver

### LPSCI DMA Driver

- [lpsci\\_status\\_t LPSCI\\_DRV\\_DmaInit](#) (uint32\_t instance, [lpsci\\_dma\\_state\\_t](#) \*lpsciDmaStatePtr, const [lpsci\\_dma\\_user\\_config\\_t](#) \*lpsciUserConfig)  
*Initializes an LPSCI instance to work with DMA.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_DmaDeinit](#) (uint32\_t instance)  
*Shuts down the LPSCI.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_DmaSendDataBlocking](#) (uint32\_t instance, const uint8\_t \*txBuff, uint32\_t txSize, uint32\_t timeout)  
*Sends (transmits) data out through the LPSCI-DMA module using blocking method.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_DmaSendData](#) (uint32\_t instance, const uint8\_t \*txBuff, uint32\_t txSize)  
*Sends (transmits) data through the LPSCI-DMA module using a non-blocking method.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_DmaGetTransmitStatus](#) (uint32\_t instance, uint32\_t \*bytesRemaining)  
*Returns whether the previous LPSCI-DMA transmit has finished.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_DmaAbortSendingData](#) (uint32\_t instance)  
*Terminates a non-blocking LPSCI-DMA transmission early.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_DmaReceiveDataBlocking](#) (uint32\_t instance, uint8\_t \*rxBuff, uint32\_t rxSize, uint32\_t timeout)  
*Gets (receives) data from the LPSCI-DMA module using a blocking method.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_DmaReceiveData](#) (uint32\_t instance, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Gets (receives) data from the LPSCI-DMA module using a non-blocking method.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_DmaGetReceiveStatus](#) (uint32\_t instance, uint32\_t \*bytesRemaining)  
*Returns whether the previous LPSCI-DMA receive is complete.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_DmaAbortReceivingData](#) (uint32\_t instance)  
*Terminates a non-blocking LPSCI-DMA receive early.*

### LPSCI Interrupt Driver

- [lpsci\\_status\\_t LPSCI\\_DRV\\_Init](#) (uint32\_t instance, [lpsci\\_state\\_t](#) \*lpsciStatePtr, const [lpsci\\_user\\_config\\_t](#) \*lpsciUserConfig)  
*Initializes an LPSCI instance for operation.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_Deinit](#) (uint32\_t instance)  
*Shuts down the LPSCI by disabling interrupts and the transmitter/receiver.*
- [lpsci\\_rx\\_callback\\_t LPSCI\\_DRV\\_InstallRxCallback](#) (uint32\_t instance, [lpsci\\_rx\\_callback\\_t](#) function, uint8\_t \*rxBuff, void \*callbackParam, bool alwaysEnableRxIrq)  
*Installs callback function for the LPSCI receive.*
- [lpsci\\_tx\\_callback\\_t LPSCI\\_DRV\\_InstallTxCallback](#) (uint32\_t instance, [lpsci\\_tx\\_callback\\_t](#) function, uint8\_t \*txBuff, void \*callbackParam)  
*Installs callback function for the LPSCI transmit.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_SendDataBlocking](#) (uint32\_t instance, const uint8\_t \*txBuff, uint32\_t txSize, uint32\_t timeout)  
*Sends (transmits) data out through the LPSCI module using a blocking method.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_SendData](#) (uint32\_t instance, const uint8\_t \*txBuff, uint32\_t txSize)  
*Sends (transmits) data through the LPSCI module using a non-blocking method.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_GetTransmitStatus](#) (uint32\_t instance, uint32\_t \*bytesRemaining)  
*Returns whether the previous LPSCI transmit has finished.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_AbortSendingData](#) (uint32\_t instance)  
*Terminates an asynchronous LPSCI transmission early.*

- [lpsci\\_status\\_t LPSCI\\_DRV\\_ReceiveDataBlocking](#) (uint32\_t instance, uint8\_t \*rxBuff, uint32\_t rxSize, uint32\_t timeout)  
*Gets (receives) data from the LPSCI module using a blocking method.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_ReceiveData](#) (uint32\_t instance, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Gets (receives) data from the LPSCI module using a non-blocking method.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_GetReceiveStatus](#) (uint32\_t instance, uint32\_t \*bytesRemaining)  
*Returns whether the previous LPSCI receive is complete.*
- [lpsci\\_status\\_t LPSCI\\_DRV\\_AbortReceivingData](#) (uint32\_t instance)  
*Terminates an asynchronous LPSCI receive early.*

### 24.3.6 Data Structure Documentation

#### 24.3.6.1 struct lpsci\_dma\_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.*
- [dma\\_channel\\_t dmaLpsciTx](#)  
*DMA channel used for send.*
- [dma\\_channel\\_t dmaLpsciRx](#)  
*DMA channel used for receive.*

## LPSCI Peripheral driver

### 24.3.6.1.0.39 Field Documentation

- 24.3.6.1.0.39.1 `volatile bool lpsci_dma_state_t::isTxBusy`
- 24.3.6.1.0.39.2 `volatile bool lpsci_dma_state_t::isRxBusy`
- 24.3.6.1.0.39.3 `volatile bool lpsci_dma_state_t::isTxBlocking`
- 24.3.6.1.0.39.4 `volatile bool lpsci_dma_state_t::isRxBlocking`
- 24.3.6.1.0.39.5 `semaphore_t lpsci_dma_state_t::txIrqSync`
- 24.3.6.1.0.39.6 `semaphore_t lpsci_dma_state_t::rxIrqSync`
- 24.3.6.1.0.39.7 `dma_channel_t lpsci_dma_state_t::dmaLpSciTx`
- 24.3.6.1.0.39.8 `dma_channel_t lpsci_dma_state_t::dmaLpSciRx`

### 24.3.6.2 struct `lpsci_dma_user_config_t`

Use an instance of this structure with the `LPSCI_DRV_Init()` function. This enables configuration of the most common settings of the LPSCI peripheral with a single function call. Settings include: LPSCI baud rate, LPSCI parity mode: disabled (default), or even or odd, the number of stop bits, and the number of bits per data word.

#### Data Fields

- `clock_lpsci_src_t clockSource`  
*LPSCI clock source in `fsl_sim_hal_<device>.h`.*
- `uint32_t baudRate`  
*LPSCI baud rate.*
- `lpsci_parity_mode_t parityMode`  
*parity mode, disabled (default), even, odd*
- `lpsci_stop_bit_count_t stopBitCount`  
*number of stop bits, 1 stop bit (default) or 2 stop bits*
- `lpsci_bit_count_per_char_t bitCountPerChar`  
*number of bits, 8-bit (default) or 9-bit in a word (up to 10-bits in some LPSCI instances)*

### 24.3.6.3 struct `lpsci_state_t`

This structure holds data used by the LPSCI 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. The LPSCI driver populates the members.

#### 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.*

  - `lpsci_rx_callback_t rxCallback`
- Callback to invoke after receiving byte.*

  - `void * rxCallbackParam`
- Receive callback parameter pointer.*

  - `lpsci_tx_callback_t txCallback`
- Callback to invoke after transmitting byte.*

  - `void * txCallbackParam`
- Transmit callback parameter pointer.*

## LPSCI Peripheral driver

### 24.3.6.3.0.40 Field Documentation

- 24.3.6.3.0.40.1 `const uint8_t* lpsci_state_t::txBuff`
- 24.3.6.3.0.40.2 `uint8_t* lpsci_state_t::rxBuff`
- 24.3.6.3.0.40.3 `volatile size_t lpsci_state_t::txSize`
- 24.3.6.3.0.40.4 `volatile size_t lpsci_state_t::rxSize`
- 24.3.6.3.0.40.5 `volatile bool lpsci_state_t::isTxBusy`
- 24.3.6.3.0.40.6 `volatile bool lpsci_state_t::isRxBusy`
- 24.3.6.3.0.40.7 `volatile bool lpsci_state_t::isTxBlocking`
- 24.3.6.3.0.40.8 `volatile bool lpsci_state_t::isRxBlocking`
- 24.3.6.3.0.40.9 `semaphore_t lpsci_state_t::txIrqSync`
- 24.3.6.3.0.40.10 `semaphore_t lpsci_state_t::rxIrqSync`
- 24.3.6.3.0.40.11 `lpsci_rx_callback_t lpsci_state_t::rxCallback`
- 24.3.6.3.0.40.12 `void* lpsci_state_t::rxCallbackParam`
- 24.3.6.3.0.40.13 `lpsci_tx_callback_t lpsci_state_t::txCallback`
- 24.3.6.3.0.40.14 `void* lpsci_state_t::txCallbackParam`

### 24.3.6.4 struct `lpsci_user_config_t`

#### Data Fields

- `clock_lpsci_src_t` `clockSource`  
*LPSCI clock source in `fsl_sim_hal_'device'.h`.*
- `uint32_t` `baudRate`  
*LPSCI baud rate.*
- `lpsci_parity_mode_t` `parityMode`  
*parity mode, disabled (default), even, odd*
- `lpsci_stop_bit_count_t` `stopBitCount`  
*number of stop bits, 1 stop bit (default) or 2 stop bits*
- `lpsci_bit_count_per_char_t` `bitCountPerChar`  
*number of bits, 8-bit (default) or 9-bit in a word (up to 10-bits in some LPSCI instances)*



### 24.3.7 Typedef Documentation

#### 24.3.7.1 typedef void(\* lpsci\_rx\_callback\_t)(uint32\_t instance, void \*lpsciState)

### 24.3.8 Function Documentation

#### 24.3.8.1 lpsci\_status\_t LPSCI\_DRV\_DmaInit ( uint32\_t instance, lpsci\_dma\_state\_t \* lpsciDmaStatePtr, const lpsci\_dma\_user\_config\_t \* lpsciUserConfig )

This function initializes the run-time state structure to keep track of the on-going transfers, un-gates the clock to the LPSCI 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, the LPSCI module transmitter and receiver. This example shows how to set up the `lpsci_dma_state_t` and the `lpsci_user_config_t` parameters and how to call the `LPSCI_DRV_DmaInit` function by passing in these parameters:

```
lpsci_user_config_t lpsciConfig;
lpsciConfig.baudRate = 9600;
lpsciConfig.bitCountPerChar = kLpsci8BitsPerChar;
lpsciConfig.parityMode = kLpsciParityDisabled;
lpsciConfig.stopBitCount = kLpsciOneStopBit;
lpsci_dma_state_t lpsciDmaState;
LPSCI_DRV_DmaInit(instance, &lpsciDmaState, &lpsciConfig);
```

#### Parameters

|                          |                                                                                                                                                                                                                                              |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>          | The LPSCI instance number.                                                                                                                                                                                                                   |
| <i>lpsciDmaState-Ptr</i> | A pointer to the LPSCI driver state structure memory. The user passes in the memory for the run-time state structure. The LPSCI driver populates the members. This run-time state structure keeps track of the current transfer in progress. |
| <i>lpsciUser-Config</i>  | The user configuration structure of type <code>lpsci_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_LPSCI_Success`.

#### 24.3.8.2 lpsci\_status\_t LPSCI\_DRV\_DmaDeinit ( uint32\_t instance )

This function disables the LPSCI-DMA trigger and disables the transmitter and receiver.

## LPSCI Peripheral driver

### Parameters

|                 |                            |
|-----------------|----------------------------|
| <i>instance</i> | The LPSCI instance number. |
|-----------------|----------------------------|

### Returns

An error code or kStatus\_LPSCI\_Success.

#### 24.3.8.3 `lpsci_status_t LPSCI_DRV_DmaSendDataBlocking ( uint32_t instance, const uint8_t * txBuff, uint32_t txSize, uint32_t timeout )`

### Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>instance</i> | The LPSCI 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\_LPSCI\_Success.

#### 24.3.8.4 `lpsci_status_t LPSCI_DRV_DmaSendData ( uint32_t instance, const uint8_t * txBuff, uint32_t txSize )`

### Parameters

|                 |                                                                     |
|-----------------|---------------------------------------------------------------------|
| <i>instance</i> | The LPSCI 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\_LPSCI\_Success.

#### 24.3.8.5 `lpsci_status_t LPSCI_DRV_DmaGetTransmitStatus ( uint32_t instance, uint32_t * bytesRemaining )`

Parameters

|                        |                                                                                                            |
|------------------------|------------------------------------------------------------------------------------------------------------|
| <i>instance</i>        | The LPSCI module base address.                                                                             |
| <i>bytes-Remaining</i> | A pointer to a value that is populated with the number of bytes that are remaining in the active transfer. |

Returns

An error code or `kStatus_LPSCI_Success`.

Return values

|                              |                                                                                                                                       |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_LPSCI_Success</i> | The transmit has completed successfully.                                                                                              |
| <i>kStatus_LPSCI_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. |

**24.3.8.6 `lpsci_status_t LPSCI_DRV_DmaAbortSendingData ( uint32_t instance )`**

Parameters

|                 |                                |
|-----------------|--------------------------------|
| <i>instance</i> | The LPSCI module base address. |
|-----------------|--------------------------------|

Returns

An error code or `kStatus_LPSCI_Success`.

Return values

|                                            |                                           |
|--------------------------------------------|-------------------------------------------|
| <i>kStatus_LPSCI_Success</i>               | The transmit was successful.              |
| <i>kStatus_LPSCI_No-TransmitInProgress</i> | No transmission is currently in progress. |

**24.3.8.7 `lpsci_status_t LPSCI_DRV_DmaReceiveDataBlocking ( uint32_t instance, uint8_t * rxBuff, uint32_t rxSize, uint32_t timeout )`**

## LPSCI Peripheral driver

### Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>instance</i> | The LPSCI 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\_LPSCI\_Success.

#### 24.3.8.8 `lpsc_i_status_t LPSCI_DRV_DmaReceiveData ( uint32_t instance, uint8_t * rxBuff, uint32_t rxSize )`

### Parameters

|                 |                                                                    |
|-----------------|--------------------------------------------------------------------|
| <i>instance</i> | The LPSCI 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\_LPSCI\_Success.

#### 24.3.8.9 `lpsc_i_status_t LPSCI_DRV_DmaGetReceiveStatus ( uint32_t instance, uint32_t * bytesRemaining )`

### Parameters

|                        |                                                                                                                         |
|------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>        | The LPSCI module base address.                                                                                          |
| <i>bytes-Remaining</i> | A pointer to a value that is populated with the number of bytes which still need to be received in the active transfer. |

### Returns

An error code or kStatus\_LPSCI\_Success.

Return values

|                              |                                                                                                                                |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_LPSCI_Success</i> | The receive has completed successfully.                                                                                        |
| <i>kStatus_LPSCI_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. |

**24.3.8.10 lpsci\_status\_t LPSCI\_DRV\_DmaAbortReceivingData ( uint32\_t instance )**

Parameters

|                 |                                |
|-----------------|--------------------------------|
| <i>instance</i> | The LPSCI module base address. |
|-----------------|--------------------------------|

Returns

An error code or *kStatus\_LPSCI\_Success*.

Return values

|                                            |                                      |
|--------------------------------------------|--------------------------------------|
| <i>kStatus_LPSCI_Success</i>               | The receive was successful.          |
| <i>kStatus_LPSCI_No-TransmitInProgress</i> | No receive is currently in progress. |

**24.3.8.11 lpsci\_status\_t LPSCI\_DRV\_Init ( uint32\_t instance, lpsci\_state\_t \* lpsciStatePtr, const lpsci\_user\_config\_t \* lpsciUserConfig )**

This function initializes the run-time state structure to keep track of the on-going transfers, un-gates the clock to the LPSCI 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 LPSCI module transmitter and receiver. This example shows how to set up the *lpsci\_state\_t* and the *lpsci\_user\_config\_t* parameters and how to call the *LPSCI\_DRV\_Init* function by passing in these parameters:

```
lpsci_user_config_t lpsciConfig;
lpsciConfig.clockSource = kClockLpisciSrcPllFllSel;
lpsciConfig.baudRate = 9600;
lpsciConfig.bitCountPerChar = kLpisci8BitsPerChar;
lpsciConfig.parityMode = kLpisciParityDisabled;
lpsciConfig.stopBitCount = kLpisciOneStopBit;
lpsci_state_t lpsciState;
LPSCI_DRV_Init(instance, &lpsciState, &lpsciConfig);
```

## LPSCI Peripheral driver

### Parameters

|                         |                                                                                                                                                                                                                                             |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>         | The LPSCI instance number.                                                                                                                                                                                                                  |
| <i>lpsciStatePtr</i>    | A pointer to the LPSCI driver state structure memory. The user passes in the memory for the run-time state structure. The LPSCI driver populates the members. The run-time state structure keeps track of the current transfer in progress. |
| <i>lpsciUser-Config</i> | The user configuration structure of type <a href="#">lpsci_user_config_t</a> . The user populates the members of this structure and passes the pointer of the structure to the function.                                                    |

### Returns

An error code or `kStatus_LPSCI_Success`.

### 24.3.8.12 `lpsci_status_t LPSCI_DRV_Deinit ( uint32_t instance )`

This function disables the LPSCI interrupts, disables the transmitter and receiver, and flushes the FIFOs (for modules that support FIFOs).

### Parameters

|                 |                            |
|-----------------|----------------------------|
| <i>instance</i> | The LPSCI instance number. |
|-----------------|----------------------------|

### Returns

An error code or `kStatus_LPSCI_Success`.

### 24.3.8.13 `lpsci_rx_callback_t LPSCI_DRV_InstallRxCallback ( uint32_t instance, lpsci_rx_callback_t function, uint8_t * rxBuff, void * callbackParam, bool alwaysEnableRxIrq )`

### Parameters

|                 |                                                                                                            |
|-----------------|------------------------------------------------------------------------------------------------------------|
| <i>instance</i> | The LPSCI instance number.                                                                                 |
| <i>function</i> | The LPSCI receive callback function.                                                                       |
| <i>rxBuff</i>   | The receive buffer used inside IRQHandler. This buffer must be kept as long as the callback is functional. |

|                           |                                               |
|---------------------------|-----------------------------------------------|
| <i>callbackParam</i>      | The LPSCI receive callback parameter pointer. |
| <i>alwaysEnable-RxIrq</i> | Whether always enable receive IRQ or not.     |

Returns

Former LPSCI receive callback function pointer.

**24.3.8.14** `lpsci_tx_callback_t LPSCI_DRV_InstallTxCallback ( uint32_t instance, lpsci_tx_callback_t function, uint8_t * txBuff, void * callbackParam )`

Note

After the callback is installed, it bypasses part of the LPSCI IRQHandler logic. Therefore, the callback needs to handle the indexes of *txBuff* and *txSize*.

Parameters

|                      |                                                                                                        |
|----------------------|--------------------------------------------------------------------------------------------------------|
| <i>instance</i>      | The LPSCI instance number.                                                                             |
| <i>function</i>      | The LPSCI transmit callback function.                                                                  |
| <i>txBuff</i>        | The transmit buffer used inside IRQHandler. This buffer must be kept as long as the callback is alive. |
| <i>callbackParam</i> | The LPSCI transmit callback parameter pointer.                                                         |

Returns

Former LPSCI transmit callback function pointer.

**24.3.8.15** `lpsci_status_t LPSCI_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 sends data through the LPSCI port.

Parameters

---

## LPSCI Peripheral driver

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>instance</i> | The LPSCI 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\_LPSCI\_Success.

#### 24.3.8.16 `lpsci_status_t LPSCI_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 LPSCI to perform a full duplex operation (simultaneously transmit and receive).

### Parameters

|                 |                                                                     |
|-----------------|---------------------------------------------------------------------|
| <i>instance</i> | The LPSCI 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\_LPSCI\_Success.

#### 24.3.8.17 `lpsci_status_t LPSCI_DRV_GetTransmitStatus ( uint32_t instance, uint32_t * bytesRemaining )`

When performing an a-sync 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



|                        |                                                                                                            |
|------------------------|------------------------------------------------------------------------------------------------------------|
| <i>instance</i>        | The LPSCI 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

Current transmission status.

Return values

|                              |                                                                                                                                     |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_LPSCI_Success</i> | The transmit has completed successfully.                                                                                            |
| <i>kStatus_LPSCI_TxBusy</i>  | The transmit is still in progress. <i>bytesRemaining</i> is filled with the number of bytes which are transmitted up to that point. |

#### 24.3.8.18 `lpsci_status_t LPSCI_DRV_AbortSendingData ( uint32_t instance )`

During an a-sync LPSCI transmission, the user can terminate the transmission early if the transmission is still in progress.

Parameters

|                 |                                |
|-----------------|--------------------------------|
| <i>instance</i> | The LPSCI module base address. |
|-----------------|--------------------------------|

Returns

Whether the aborting was successful or not.

Return values

|                                            |                                           |
|--------------------------------------------|-------------------------------------------|
| <i>kStatus_LPSCI_Success</i>               | The transmit was successful.              |
| <i>kStatus_LPSCI_No-TransmitInProgress</i> | No transmission is currently in progress. |

#### 24.3.8.19 `lpsci_status_t LPSCI_DRV_ReceiveDataBlocking ( uint32_t instance, uint8_t * rxBuff, uint32_t rxSize, uint32_t timeout )`

A blocking (also known as synchronous) function does not return until the receive is complete. This blocking function sends data through the LPSCI port.

## LPSCI Peripheral driver

### Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>instance</i> | The LPSCI 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\_LPSCI\_Success.

#### 24.3.8.20 `lpsci_status_t LPSCI_DRV_ReceiveData ( uint32_t instance, uint8_t * rxBuff, uint32_t rxSize )`

A non-blocking (also known as synchronous) function returns immediately after initiating the receive function. The application has to get the receive status to see when the receive is complete. The asynchronous method of transmitting and receiving allows the LPSCI to perform a full duplex operation (simultaneously transmit and receive).

### Parameters

|                 |                                                                    |
|-----------------|--------------------------------------------------------------------|
| <i>instance</i> | The LPSCI 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\_LPSCI\_Success.

#### 24.3.8.21 `lpsci_status_t LPSCI_DRV_GetReceiveStatus ( uint32_t instance, uint32_t * bytesRemaining )`

When performing an a-sync receive, call this function to ascertain the state of the current receive progress: in progress (or busy) or complete (success). If the receive is still in progress, the user can obtain the number of words that have been received.

Parameters

|                        |                                                                                                                         |
|------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>        | The LPSCI module base address.                                                                                          |
| <i>bytes-Remaining</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

Current receive status.

Return values

|                              |                                                                                                                                 |
|------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_LPSCI_Success</i> | The receive has completed successfully.                                                                                         |
| <i>kStatus_LPSCI_RxBusy</i>  | The receive is still in progress. <i>bytesRemaining</i> is filled with the number of bytes which are received up to that point. |

**24.3.8.22 lpsci\_status\_t LPSCI\_DRV\_AbortReceivingData ( uint32\_t instance )**

During an a-sync LPSCI receive, the user can terminate the receive early if the receive is still in progress.

Parameters

|                 |                                |
|-----------------|--------------------------------|
| <i>instance</i> | The LPSCI module base address. |
|-----------------|--------------------------------|

Returns

Whether the action success or not.

Return values

|                                            |                                      |
|--------------------------------------------|--------------------------------------|
| <i>kStatus_LPSCI_Success</i>               | The receive was successful.          |
| <i>kStatus_LPSCI_No-TransmitInProgress</i> | No receive is currently in progress. |

**24.3.9 Variable Documentation**

**24.3.9.1 UART0\_Type\* const g\_lpsciBase[UART0\_INSTANCE\_COUNT]**

**24.3.9.2 UART0\_Type\* const g\_lpsciBase[UART0\_INSTANCE\_COUNT]**



## Chapter 25

### Low Power Timer (LPTMR)

#### 25.1 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](#)

## LPTMR HAL driver

### 25.2 LPTMR HAL driver

#### 25.2.1 Overview

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

#### Data Structures

- struct [lptmr\\_prescaler\\_user\\_config\\_t](#)  
*Define LPTMR prescaler user configure. [More...](#)*
- struct [lptmr\\_working\\_mode\\_user\\_config\\_t](#)  
*Define LPTMR working mode user configure. [More...](#)*

#### Enumerations

- enum [lptmr\\_pin\\_select\\_t](#) {  
[kLptmrPinSelectInput0](#) = 0x0U,  
[kLptmrPinSelectInput1](#) = 0x1U,  
[kLptmrPinSelectInput2](#) = 0x2U,  
[kLptmrPinSelectInput3](#) = 0x3U }  
*LPTMR pin selection, used in pulse counter mode.*
- enum [lptmr\\_pin\\_polarity\\_t](#) {  
[kLptmrPinPolarityActiveHigh](#) = 0x0U,  
[kLptmrPinPolarityActiveLow](#) = 0x1U }  
*LPTMR pin polarity, used in pulse counter mode.*
- enum [lptmr\\_timer\\_mode\\_t](#) {  
[kLptmrTimerModeTimeCounter](#) = 0x0U,  
[kLptmrTimerModePulseCounter](#) = 0x1U }  
*LPTMR timer mode selection.*
- enum [lptmr\\_prescaler\\_value\\_t](#) {  
[kLptmrPrescalerDivide2](#) = 0x0U,  
[kLptmrPrescalerDivide4GlitchFilter2](#) = 0x1U,  
[kLptmrPrescalerDivide8GlitchFilter4](#) = 0x2U,  
[kLptmrPrescalerDivide16GlitchFilter8](#) = 0x3U,  
[kLptmrPrescalerDivide32GlitchFilter16](#) = 0x4U,  
[kLptmrPrescalerDivide64GlitchFilter32](#) = 0x5U,  
[kLptmrPrescalerDivide128GlitchFilter64](#) = 0x6U,  
[kLptmrPrescalerDivide256GlitchFilter128](#) = 0x7U,  
[kLptmrPrescalerDivide512GlitchFilter256](#) = 0x8U,  
[kLptmrPrescalerDivide1024GlitchFilter512](#) = 0x9U,  
[kLptmrPrescalerDivide2048GlitchFilter1024](#) = 0xAU,  
[kLptmrPrescalerDivide4096GlitchFilter2048](#) = 0xBU,  
[kLptmrPrescalerDivide8192GlitchFilter4096](#) = 0xCU,  
[kLptmrPrescalerDivide16384GlitchFilter8192](#) = 0xDU,  
[kLptmrPrescalerDivide32768GlitchFilter16384](#) = 0xEU,

- `kLptmrPrescalerDivide65536GlitchFilter32768 = 0xFU }`
- LPTMR prescaler value.*
- enum `lptmr_prescaler_clock_select_t` {
  - `kLptmrPrescalerClock0 = 0x0U,`
  - `kLptmrPrescalerClock1 = 0x1U,`
  - `kLptmrPrescalerClock2 = 0x2U,`
  - `kLptmrPrescalerClock3 = 0x3U }`*LPTMR prescaler/glitch filter clock select.*
- 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` (`LPTMR_Type *base`)
  - Enables the LPTMR module operation.*
- static void `LPTMR_HAL_Disable` (`LPTMR_Type *base`)
  - Disables the LPTMR module operation.*
- static void `LPTMR_HAL_ClearIntFlag` (`LPTMR_Type *base`)
  - Clears the LPTMR interrupt flag if set.*
- static bool `LPTMR_HAL_IsIntPending` (`LPTMR_Type *base`)
  - Returns the current LPTMR interrupt flag.*
- static void `LPTMR_HAL_SetIntCmd` (`LPTMR_Type *baseAddr`, bool enable)
  - Enables or disables the LPTMR interrupt.*
- void `LPTMR_HAL_SetTimerWorkingMode` (`LPTMR_Type *base`, `lptmr_working_mode_user_config_t` timerMode)
  - Configures the LPTMR working mode.*
- void `LPTMR_HAL_SetPrescalerMode` (`LPTMR_Type *base`, `lptmr_prescaler_user_config_t` prescaler\_config)
  - Sets the LPTMR prescaler mode.*
- static void `LPTMR_HAL_SetCompareValue` (`LPTMR_Type *base`, `uint32_t` compareValue)
  - Sets the LPTMR compare value.*
- static `uint32_t` `LPTMR_HAL_GetCompareValue` (`LPTMR_Type *base`)
  - Gets the LPTMR compare value.*
- `uint32_t` `LPTMR_HAL_GetCounterValue` (`LPTMR_Type *base`)
  - Gets the LPTMR counter value.*
- void `LPTMR_HAL_Init` (`LPTMR_Type *base`)
  - Restores the LPTMR module to reset state.*

## LPTMR HAL driver

### 25.2.2 Data Structure Documentation

#### 25.2.2.1 struct `lptmr_prescaler_user_config_t`

##### Data Fields

- bool [prescalerBypass](#)  
*Set this value will by pass the Prescaler or glitch filter.*
- [lptmr\\_prescaler\\_clock\\_select\\_t](#) `prescalerClockSelect`  
*Selects the clock to be used by the LPTMR prescaler/glitch filter.*
- [lptmr\\_prescaler\\_value\\_t](#) `prescalerValue`  
*Configures the size of the Prescaler in Time Counter mode or width of the glitch filter in Pulse Counter mode.*

##### 25.2.2.1.0.41 Field Documentation

25.2.2.1.0.41.1 bool `lptmr_prescaler_user_config_t::prescalerBypass`

25.2.2.1.0.41.2 [lptmr\\_prescaler\\_clock\\_select\\_t](#) `lptmr_prescaler_user_config_t::prescalerClockSelect`

25.2.2.1.0.41.3 [lptmr\\_prescaler\\_value\\_t](#) `lptmr_prescaler_user_config_t::prescalerValue`

#### 25.2.2.2 struct `lptmr_working_mode_user_config_t`

##### Data Fields

- [lptmr\\_timer\\_mode\\_t](#) `timerModeSelect`  
*Selects the LPTMR working mode: Timer or Pulse Counter.*
- bool [freeRunningEnable](#)  
*Enables or disables the LPTMR free running.*
- [lptmr\\_pin\\_polarity\\_t](#) `pinPolarity`  
*Specifies LPTMR pulse input pin polarity.*
- [lptmr\\_pin\\_select\\_t](#) `pinSelect`  
*Specifies LPTMR pulse input pin select.*



#### 25.2.2.2.0.42 Field Documentation

25.2.2.2.0.42.1 `lptmr_timer_mode_t` `lptmr_working_mode_user_config_t::timerModeSelect`

25.2.2.2.0.42.2 `bool` `lptmr_working_mode_user_config_t::freeRunningEnable`

25.2.2.2.0.42.3 `lptmr_pin_polarity_t` `lptmr_working_mode_user_config_t::pinPolarity`

25.2.2.2.0.42.4 `lptmr_pin_select_t` `lptmr_working_mode_user_config_t::pinSelect`

### 25.2.3 Enumeration Type Documentation

#### 25.2.3.1 enum `lptmr_pin_select_t`

Enumerator

*`kLptmrPinSelectInput0`* Pulse counter input 0 is selected.  
*`kLptmrPinSelectInput1`* Pulse counter input 1 is selected.  
*`kLptmrPinSelectInput2`* Pulse counter input 2 is selected.  
*`kLptmrPinSelectInput3`* Pulse counter input 3 is selected.

#### 25.2.3.2 enum `lptmr_pin_polarity_t`

Enumerator

*`kLptmrPinPolarityActiveHigh`* Pulse Counter input source is active-high.  
*`kLptmrPinPolarityActiveLow`* Pulse Counter input source is active-low.

#### 25.2.3.3 enum `lptmr_timer_mode_t`

Enumerator

*`kLptmrTimerModeTimeCounter`* Time Counter mode.  
*`kLptmrTimerModePulseCounter`* Pulse Counter mode.

#### 25.2.3.4 enum `lptmr_prescaler_value_t`

Enumerator

*`kLptmrPrescalerDivide2`* Prescaler divide 2, glitch filter invalid.  
*`kLptmrPrescalerDivide4GlitchFilter2`* Prescaler divide 4, glitch filter 2.  
*`kLptmrPrescalerDivide8GlitchFilter4`* Prescaler divide 8, glitch filter 4.  
*`kLptmrPrescalerDivide16GlitchFilter8`* Prescaler divide 16, glitch filter 8.  
*`kLptmrPrescalerDivide32GlitchFilter16`* Prescaler divide 32, glitch filter 16.

## LPTMR HAL driver

*kLptmrPrescalerDivide64GlitchFilter32* Prescaler divide 64, glitch filter 32.  
*kLptmrPrescalerDivide128GlitchFilter64* Prescaler divide 128, glitch filter 64.  
*kLptmrPrescalerDivide256GlitchFilter128* Prescaler divide 256, glitch filter 128.  
*kLptmrPrescalerDivide512GlitchFilter256* Prescaler divide 512, glitch filter 256.  
*kLptmrPrescalerDivide1024GlitchFilter512* Prescaler divide 1024, glitch filter 512.  
*kLptmrPrescalerDivide2048GlitchFilter1024* Prescaler divide 2048 glitch filter 1024.  
*kLptmrPrescalerDivide4096GlitchFilter2048* Prescaler divide 4096, glitch filter 2048.  
*kLptmrPrescalerDivide8192GlitchFilter4096* Prescaler divide 8192, glitch filter 4096.  
*kLptmrPrescalerDivide16384GlitchFilter8192* Prescaler divide 16384, glitch filter 8192.  
*kLptmrPrescalerDivide32768GlitchFilter16384* Prescaler divide 32768, glitch filter 16384.  
*kLptmrPrescalerDivide65536GlitchFilter32768* Prescaler divide 65536, glitch filter 32768.

### 25.2.3.5 enum lptmr\_prescaler\_clock\_select\_t

Enumerator

*kLptmrPrescalerClock0* Prescaler/glitch filter clock 0 selected.  
*kLptmrPrescalerClock1* Prescaler/glitch filter clock 1 selected.  
*kLptmrPrescalerClock2* Prescaler/glitch filter clock 2 selected.  
*kLptmrPrescalerClock3* Prescaler/glitch filter clock 3 selected.

### 25.2.3.6 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 cannot be called in time counter mode.  
*kStatus\_LPTMR\_InvalidInPulseCounterMode* Function cannot be 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.

## 25.2.4 Function Documentation

25.2.4.1 `static void LPTMR_HAL_Enable ( LPTMR_Type * base ) [inline], [static]`

Parameters

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

**25.2.4.2 static void LPTMR\_HAL\_Disable ( LPTMR\_Type \* *base* ) [inline], [static]**

Parameters

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

**25.2.4.3 static void LPTMR\_HAL\_ClearIntFlag ( LPTMR\_Type \* *base* ) [inline], [static]**

Parameters

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

**25.2.4.4 static bool LPTMR\_HAL\_IsIntPending ( LPTMR\_Type \* *base* ) [inline], [static]**

Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>base</i> | The LPTMR peripheral base address |
|-------------|-----------------------------------|

Returns

State of LPTMR interrupt flag

Return values

|              |                          |
|--------------|--------------------------|
| <i>true</i>  | An interrupt is pending. |
| <i>false</i> | No interrupt is pending. |

**25.2.4.5 static void LPTMR\_HAL\_SetIntCmd ( LPTMR\_Type \* *baseAddr*, bool *enable* ) [inline], [static]**

## LPTMR HAL driver

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>baseAddr</i> | The LPTMR peripheral base address   |
| <i>enable</i>   | Pass true to enable LPTMR interrupt |

**25.2.4.6 void LPTMR\_HAL\_SetTimerWorkingMode ( LPTMR\_Type \* *base*,  
lptmr\_working\_mode\_user\_config\_t *timerMode* )**

Parameters

|                  |                                                                                              |
|------------------|----------------------------------------------------------------------------------------------|
| <i>base</i>      | The LPTMR peripheral base address.                                                           |
| <i>timerMode</i> | Specifies LPTMR working mode configure, see <a href="#">lptmr_working_mode_user_config_t</a> |

**25.2.4.7 void LPTMR\_HAL\_SetPrescalerMode ( LPTMR\_Type \* *base*,  
lptmr\_prescaler\_user\_config\_t *prescaler\_config* )**

Parameters

|                                    |                                                                                        |
|------------------------------------|----------------------------------------------------------------------------------------|
| <i>base</i>                        | The LPTMR peripheral base address.                                                     |
| <i>prescaler_</i><br><i>config</i> | Specifies LPTMR prescaler configure, see <a href="#">lptmr_prescaler_user_config_t</a> |

**25.2.4.8 static void LPTMR\_HAL\_SetCompareValue ( LPTMR\_Type \* *base*, uint32\_t  
*compareValue* ) [inline], [static]**

Parameters

|                     |                                                  |
|---------------------|--------------------------------------------------|
| <i>base</i>         | The LPTMR peripheral base address.               |
| <i>compareValue</i> | Specifies LPTMR compare value, less than 0xFFFFU |

**25.2.4.9 static uint32\_t LPTMR\_HAL\_GetCompareValue ( LPTMR\_Type \* *base* )  
[inline], [static]**

Parameters

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

#### 25.2.4.10 uint32\_t LPTMR\_HAL\_GetCounterValue ( LPTMR\_Type \* *base* )

Parameters

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

Returns

Current LPTMR counter value

#### 25.2.4.11 void LPTMR\_HAL\_Init ( LPTMR\_Type \* *base* )

Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>base</i> | The LPTMR peripheral base address |
|-------------|-----------------------------------|

## LPTMR Peripheral driver

### 25.3 LPTMR Peripheral driver

#### 25.3.1 Overview

This section describes the programming interface of the LPTMR Peripheral driver. The LPTMR driver configures the LPTMR.

#### 25.3.2 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 LPTMR 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 = kLptmrPrescalerDivede2, // prelscaler 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);
```

#### 25.3.3 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++;
}
```

### 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

- LPTMR\_Type \*const [g\\_lptmrBase](#) []  
*Table of base addresses for LPTMR instances.*
- const IRQn\_Type [g\\_lptmrIrqId](#) [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, [lptmr\\_state\\_t](#) \*userStatePtr, const [lptmr\\_user\\_config\\_t](#) \*userConfigPtr)  
*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.*

## LPTMR Peripheral driver

### 25.3.4 Data Structure Documentation

#### 25.3.4.1 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.*

##### 25.3.4.1.0.43 Field Documentation

###### 25.3.4.1.0.43.1 bool lptmr\_user\_config\_t::freeRunningEnable

True means enable free running

###### 25.3.4.1.0.43.2 bool lptmr\_user\_config\_t::prescalerEnable

True means enable prescaler

#### 25.3.4.2 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.*



25.3.4.2.0.44 Field Documentation

25.3.4.2.0.44.1 `lptmr_callback_t lptmr_state_t::userCallbackFunc`

25.3.5 Function Documentation

25.3.5.1 `lptmr_status_t LPTMR_DRV_Init ( uint32_t instance, lptmr_state_t * userStatePtr, const lptmr_user_config_t * userConfigPtr )`

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

|                      |                                                                                              |
|----------------------|----------------------------------------------------------------------------------------------|
| <i>instance</i>      | The LPTMR peripheral instance number.                                                        |
| <i>userStatePtr</i>  | The pointer to the structure of the context memory, see <a href="#">lptmr_state_t</a> .      |
| <i>userConfigPtr</i> | The pointer to the LPTMR user configure structure, see <a href="#">lptmr_user_config_t</a> . |

Returns

`kStatus_LPTMR_Success` means succeed, otherwise means failed.

25.3.5.2 `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

|                 |                                       |
|-----------------|---------------------------------------|
| <i>instance</i> | The LPTMR peripheral instance number. |
|-----------------|---------------------------------------|

Returns

`kStatus_LPTMR_Success` means succeed, otherwise means failed.

25.3.5.3 `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.

## LPTMR Peripheral driver

### Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>instance</i> | The LPTMR peripheral instance number. |
|-----------------|---------------------------------------|

### Returns

kStatus\_LPTMR\_Success means success. Otherwise, means failure.

#### 25.3.5.4 `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.

#### 25.3.5.5 `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  $\mu$ s, 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.

#### 25.3.5.6 `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.

### Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>instance</i> | The LPTMR peripheral instance number. |
|-----------------|---------------------------------------|

### Returns

current time in microsecond unit.

#### 25.3.5.7 **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.

#### 25.3.5.8 **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.

#### 25.3.5.9 **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.

## LPTMR Peripheral driver

### Parameters

|                     |                                 |
|---------------------|---------------------------------|
| <i>instance</i>     | LPTMR instance ID.              |
| <i>userCallback</i> | User-defined callback function. |

### Returns

kStatus\_LPTMR\_Success means success. Otherwise, means failure.

### 25.3.5.10 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. |
|-----------------|--------------------|

## 25.3.6 Variable Documentation

### 25.3.6.1 LPTMR\_Type\* const g\_lptmrBase[]

### 25.3.6.2 const IRQn\_Type g\_lptmrIrqId[LPTMR\_INSTANCE\_COUNT]

## Chapter 26

# Low Power Universal Asynchronous Receiver/Transmitter (LPUART)

### 26.1 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

### 26.2 LPUART HAL driver

#### 26.2.1 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` {  
`kLpuartSinglewireTxdirectionIn = 0x0U,`  
`kLpuartSinglewireTxdirectionOut = 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 | LPUART_STAT_TDRE_SHIFT,`  
`kLpuartTxComplete = LPUART_STAT_REG_ID << LPUART_SHIFT | LPUART_STAT_TC_SHIFT,`  
`kLpuartRxDataRegFull = LPUART_STAT_REG_ID << LPUART_SHIFT | LPUART_STAT_RDRF_SHIFT,`  
`kLpuartIdleLineDetect = LPUART_STAT_REG_ID << LPUART_SHIFT | LPUART_STAT_IDLE_SHIFT,`  
`kLpuartRxOverrun = LPUART_STAT_REG_ID << LPUART_SHIFT | LPUART_STAT_OR_S-`

## LPUART HAL driver

```

HIFT,
kLpuartNoiseDetect = LPUART_STAT_REG_ID << LPUART_SHIFT | LPUART_STAT_NF_-
SHIFT,
kLpuartFrameErr = LPUART_STAT_REG_ID << LPUART_SHIFT | LPUART_STAT_FE_SH-
IFT,
kLpuartParityErr = LPUART_STAT_REG_ID << LPUART_SHIFT | LPUART_STAT_PF_SHI-
FT,
kLpuartLineBreakDetect = LPUART_STAT_REG_ID << LPUART_SHIFT | LPUART_STAT_-
LBKDE_SHIFT,
kLpuartRxActiveEdgeDetect = LPUART_STAT_REG_ID << LPUART_SHIFT | LPUART_ST-
AT_RXEDGIF_SHIFT,
kLpuartRxActive = LPUART_STAT_REG_ID << LPUART_SHIFT | LPUART_STAT_RAF_S-
HIFT,
kLpuartNoiseInCurrentWord = LPUART_DATA_REG_ID << LPUART_SHIFT | LPUART_D-
ATA_NOISY_SHIFT,
kLpuartParityErrInCurrentWord = LPUART_DATA_REG_ID << LPUART_SHIFT | LPUART-
_DATA_PARITYE_SHIFT }
 LPUART status flags.
• enum lpuart_interrupt_t {
 kLpuartIntLinBreakDetect = LPUART_BAUD_REG_ID << LPUART_SHIFT | LPUART_BAU-
D_LBKDIE_SHIFT,
 kLpuartIntRxActiveEdge = LPUART_BAUD_REG_ID << LPUART_SHIFT | LPUART_BAU-
D_RXEDGIE_SHIFT,
 kLpuartIntTxDataRegEmpty = LPUART_CTRL_REG_ID << LPUART_SHIFT | LPUART_CT-
RL_TIE_SHIFT,
 kLpuartIntTxComplete = LPUART_CTRL_REG_ID << LPUART_SHIFT | LPUART_CTRL_T-
CIE_SHIFT,
 kLpuartIntRxDataRegFull = LPUART_CTRL_REG_ID << LPUART_SHIFT | LPUART_CTRL-
_RIE_SHIFT,
 kLpuartIntIdleLine = LPUART_CTRL_REG_ID << LPUART_SHIFT | LPUART_CTRL_ILIE-
_SHIFT,
 kLpuartIntRxOverrun = LPUART_CTRL_REG_ID << LPUART_SHIFT | LPUART_CTRL_O-
RIE_SHIFT,
 kLpuartIntNoiseErrFlag = LPUART_CTRL_REG_ID << LPUART_SHIFT | LPUART_CTRL_-
NEIE_SHIFT,
 kLpuartIntFrameErrFlag = LPUART_CTRL_REG_ID << LPUART_SHIFT | LPUART_CTRL_-
FEIE_SHIFT,
 kLpuartIntParityErrFlag = LPUART_CTRL_REG_ID << LPUART_SHIFT | LPUART_CTRL_-
PEIE_SHIFT }
 LPUART interrupt configuration structure, default settings are 0 (disabled)

```

## LPUART Common Configurations

- void `LPUART_HAL_Init` (LPUART\_Type \*base)



- Initializes the LPUART controller to known state.*

  - static void [LPUART\\_HAL\\_SetTransmitterCmd](#) (LPUART\_Type \*base, bool enable)  
*Enable/Disable the LPUART transmitter.*
  - static bool [LPUART\\_HAL\\_GetTransmitterCmd](#) (LPUART\_Type \*base)  
*Gets the LPUART transmitter enabled/disabled configuration.*
  - static void [LPUART\\_HAL\\_SetReceiverCmd](#) (LPUART\_Type \*base, bool enable)  
*Enable/Disable the LPUART receiver.*
  - static bool [LPUART\\_HAL\\_GetReceiverCmd](#) (LPUART\_Type \*base)  
*Gets the LPUART receiver enabled/disabled configuration.*
  - [lpuart\\_status\\_t](#) [LPUART\\_HAL\\_SetBaudRate](#) (LPUART\_Type \*base, uint32\_t sourceClockInHz, uint32\_t desiredBaudRate)  
*Configures the LPUART baud rate.*
  - static void [LPUART\\_HAL\\_SetBaudRateDivisor](#) (LPUART\_Type \*base, uint32\_t baudRateDivisor)  
*Sets the LPUART baud rate modulo divisor.*
  - void [LPUART\\_HAL\\_SetBitCountPerChar](#) (LPUART\_Type \*base, [lpuart\\_bit\\_count\\_per\\_char\\_t](#) bitCountPerChar)  
*Configures the number of bits per character in the LPUART controller.*
  - void [LPUART\\_HAL\\_SetParityMode](#) (LPUART\_Type \*base, [lpuart\\_parity\\_mode\\_t](#) parityModeType)  
*Configures parity mode in the LPUART controller.*
  - static void [LPUART\\_HAL\\_SetStopBitCount](#) (LPUART\_Type \*base, [lpuart\\_stop\\_bit\\_count\\_t](#) stopBitCount)  
*Configures the number of stop bits in the LPUART controller.*
  - static uint32\_t [LPUART\\_HAL\\_GetDataRegAddr](#) (LPUART\_Type \*base)  
*Get LPUART tx/rx data register address.*

## LPUART Interrupts and DMA

- void [LPUART\\_HAL\\_SetIntMode](#) (LPUART\_Type \*base, [lpuart\\_interrupt\\_t](#) interrupt, bool enable)  
*Configures the LPUART module interrupts to enable/disable various interrupt sources.*
- bool [LPUART\\_HAL\\_GetIntMode](#) (LPUART\_Type \*base, [lpuart\\_interrupt\\_t](#) interrupt)  
*Returns whether the LPUART module interrupts is enabled/disabled.*

## LPUART Transfer Functions

- static void [LPUART\\_HAL\\_Putchar](#) (LPUART\_Type \*base, uint8\_t data)  
*Sends the LPUART 8-bit character.*
- void [LPUART\\_HAL\\_Putchar9](#) (LPUART\_Type \*base, uint16\_t data)  
*Sends the LPUART 9-bit character.*
- [lpuart\\_status\\_t](#) [LPUART\\_HAL\\_Putchar10](#) (LPUART\_Type \*base, uint16\_t data)  
*Sends the LPUART 10-bit character (Note: Feature available on select LPUART instances).*
- static void [LPUART\\_HAL\\_Getchar](#) (LPUART\_Type \*base, uint8\_t \*readData)  
*Gets the LPUART 8-bit character.*
- void [LPUART\\_HAL\\_Getchar9](#) (LPUART\_Type \*base, uint16\_t \*readData)  
*Gets the LPUART 9-bit character.*
- void [LPUART\\_HAL\\_Getchar10](#) (LPUART\_Type \*base, uint16\_t \*readData)  
*Gets the LPUART 10-bit character.*

## LPUART HAL driver

- void [LPUART\\_HAL\\_SendDataPolling](#) (LPUART\_Type \*base, const uint8\_t \*txBuff, uint32\_t txSize)  
*Send out multiple bytes of data using polling method.*
- [lpuart\\_status\\_t LPUART\\_HAL\\_ReceiveDataPolling](#) (LPUART\_Type \*base, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Receive multiple bytes of data using polling method.*

## LPUART Status Flags

- bool [LPUART\\_HAL\\_GetStatusFlag](#) (LPUART\_Type \*base, [lpuart\\_status\\_flag\\_t](#) statusFlag)  
*LPUART get status flag.*
- [lpuart\\_status\\_t LPUART\\_HAL\\_ClearStatusFlag](#) (LPUART\_Type \*base, [lpuart\\_status\\_flag\\_t](#) statusFlag)  
*LPUART clears an individual status flag (see [lpuart\\_status\\_flag\\_t](#) for list of status bits).*

## LPUART Special Feature Configurations

- static void [LPUART\\_HAL\\_SetIdleChar](#) (LPUART\_Type \*base, [lpuart\\_idle\\_char\\_t](#) idleConfig)  
*Configures the number of idle characters that must be received before the IDLE flag is set.*
- static [lpuart\\_idle\\_char\\_t LPUART\\_HAL\\_GetIdleChar](#) (LPUART\_Type \*base)  
*Gets the configuration of the number of idle characters that must be received before the IDLE flag is set.*
- static bool [LPUART\\_HAL\\_IsCurrentDataWithNoise](#) (LPUART\_Type \*base)  
*Checks whether the current data word was received with noise.*
- static bool [LPUART\\_HAL\\_IsCurrentDataWithFrameError](#) (LPUART\_Type \*base)  
*Checks whether the current data word was received with frame error.*
- static void [LPUART\\_HAL\\_SetTxSpecialChar](#) (LPUART\_Type \*base, 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](#) (LPUART\_Type \*base)  
*Checks whether the current data word was received with parity error.*
- static bool [LPUART\\_HAL\\_IsReceiveBufferEmpty](#) (LPUART\_Type \*base)  
*Checks whether the receive buffer is empty.*
- static bool [LPUART\\_HAL\\_WasPreviousReceiverStateIdle](#) (LPUART\_Type \*base)  
*Checks whether the previous BUS state was idle before this byte is received.*
- static void [LPUART\\_HAL\\_SetWaitModeOperation](#) (LPUART\_Type \*base, [lpuart\\_operation\\_config\\_t](#) mode)  
*Configures the LPUART operation in wait mode (operates or stops operations in wait mode).*
- static [lpuart\\_operation\\_config\\_t LPUART\\_HAL\\_GetWaitModeOperation](#) (LPUART\_Type \*base)  
*Gets the LPUART operation in wait mode (operates or stops operations in wait mode).*
- void [LPUART\\_HAL\\_SetLoopbackCmd](#) (LPUART\_Type \*base, bool enable)  
*Configures the LPUART loopback operation (enable/disable loopback operation)*
- void [LPUART\\_HAL\\_SetSingleWireCmd](#) (LPUART\_Type \*base, bool enable)  
*Configures the LPUART single-wire operation (enable/disable single-wire mode)*
- static void [LPUART\\_HAL\\_SetTxdirInSinglewireMode](#) (LPUART\_Type \*base, [lpuart\\_singlewire\\_txdir\\_t](#) direction)  
*Configures the LPUART transmit direction while in single-wire mode.*
- [lpuart\\_status\\_t LPUART\\_HAL\\_SetReceiverInStandbyMode](#) (LPUART\_Type \*base)  
*Places the LPUART receiver in standby mode.*

- static void **LPUART\_HAL\_PutReceiverInNormalMode** (LPUART\_Type \*base)  
*Places the LPUART receiver in a normal mode (disable standby mode operation).*
- static bool **LPUART\_HAL\_IsReceiverInStandby** (LPUART\_Type \*base)  
*Checks whether the LPUART receiver is in a standby mode.*
- static void **LPUART\_HAL\_SetReceiverWakeupMode** (LPUART\_Type \*base, **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** (LPUART\_Type \*base)  
*Gets the LPUART receiver wakeup method (idle line or addr-mark) from standby mode.*
- void **LPUART\_HAL\_SetIdleLineDetect** (LPUART\_Type \*base, 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** (LPUART\_Type \*base, **lpuart\_break\_char\_length\_t** length)  
*LPUART break character transmit length configuration.*
- static void **LPUART\_HAL\_SetBreakCharDetectLength** (LPUART\_Type \*base, **lpuart\_break\_char\_length\_t** length)  
*LPUART break character detect length configuration.*
- static void **LPUART\_HAL\_QueueBreakCharToSend** (LPUART\_Type \*base, bool enable)  
*LPUART transmit sends break character configuration.*
- static void **LPUART\_HAL\_SetMatchAddressMode** (LPUART\_Type \*base, **lpuart\_match\_config\_t** config)  
*LPUART configures match address mode control.*
- void **LPUART\_HAL\_SetMatchAddressReg1** (LPUART\_Type \*base, bool enable, uint8\_t value)  
*Configures address match register 1.*
- void **LPUART\_HAL\_SetMatchAddressReg2** (LPUART\_Type \*base, bool enable, uint8\_t value)  
*Configures address match register 2.*
- static void **LPUART\_HAL\_SetSendMsbFirstCmd** (LPUART\_Type \*base, bool enable)  
*LPUART sends the MSB first configuration.*
- static void **LPUART\_HAL\_SetReceiveResyncCmd** (LPUART\_Type \*base, bool enable)  
*LPUART enable/disable re-sync of received data configuration.*

## 26.2.2 Data Structure Documentation

### 26.2.2.1 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.*

## LPUART HAL driver

### 26.2.2.1.0.45 Field Documentation

#### 26.2.2.1.0.45.1 unsigned lpuart\_idle\_line\_config\_t::rxWakeldleDetect

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

## 26.2.3 Enumeration Type Documentation

### 26.2.3.1 enum lpuart\_status\_t

### 26.2.3.2 enum lpuart\_stop\_bit\_count\_t

Enumerator

*kLpuartOneStopBit* one stop bit  
*kLpuartTwoStopBit* two stop bits

### 26.2.3.3 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

### 26.2.3.4 enum lpuart\_bit\_count\_per\_char\_t

Enumerator

*kLpuart8BitsPerChar* 8-bit data characters  
*kLpuart9BitsPerChar* 9-bit data characters  
*kLpuart10BitsPerChar* 10-bit data characters

### 26.2.3.5 enum lpuart\_operation\_config\_t

Enumerator

*kLpuartOperates* LPUART continues to operate normally.  
*kLpuartStops* LPUART stops operation.

### 26.2.3.6 enum lpuart\_wakeup\_method\_t

Enumerator

*kLpuartIdleLineWake* Idle-line wakes the LPUART receiver from standby.

*kLpuartAddrMarkWake* Addr-mark wakes LPUART receiver from standby.

### 26.2.3.7 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.

### 26.2.3.8 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)

### 26.2.3.9 enum lpuart\_singlewire\_txdir\_t

Enumerator

*kLpuartSinglewireTxdirIn* LPUART Single Wire mode TXDIR input.

*kLpuartSinglewireTxdirOut* LPUART Single Wire mode TXDIR output.

### 26.2.3.10 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.

## LPUART HAL driver

### 26.2.3.11 enum lpuart\_ir\_tx\_pulsewidth\_t

Enumerator

*kLpuartIrThreeSixteenthsWidth* 3/16 pulse  
*kLpuartIrOneSixteenthWidth* 1/16 pulse  
*kLpuartIrOneThirtysecondsWidth* 1/32 pulse  
*kLpuartIrOneFourthWidth* 1/4 pulse

### 26.2.3.12 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

### 26.2.3.13 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.

### 26.2.3.14 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.

### 26.2.3.15 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* Rx Overrun, sets when new data is received before data is read from receive register.
- kLpuartNoiseDetect* Rx 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.

**26.2.3.16 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.

**26.2.4 Function Documentation**
**26.2.4.1 void LPUART\_HAL\_Init ( LPUART\_Type \* *base* )**

 Parameters
 

---

## LPUART HAL driver

|             |                      |
|-------------|----------------------|
| <i>base</i> | LPUART base pointer. |
|-------------|----------------------|

**26.2.4.2** `static void LPUART_HAL_SetTransmitterCmd ( LPUART_Type * base, bool enable ) [inline], [static]`

Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | LPUART base pointer.                        |
| <i>enable</i> | Enable(true) or disable(false) transmitter. |

**26.2.4.3** `static bool LPUART_HAL_GetTransmitterCmd ( LPUART_Type * base ) [inline], [static]`

Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | LPUART base pointer |
|-------------|---------------------|

Returns

State of LPUART transmitter enable(true)/disable(false)

**26.2.4.4** `static void LPUART_HAL_SetReceiverCmd ( LPUART_Type * base, bool enable ) [inline], [static]`

Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>base</i>   | LPUART base pointer                      |
| <i>enable</i> | Enable(true) or disable(false) receiver. |

**26.2.4.5** `static bool LPUART_HAL_GetReceiverCmd ( LPUART_Type * base ) [inline], [static]`



Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | LPUART base pointer |
|-------------|---------------------|

Returns

State of LPUART receiver enable(true)/disable(false)

**26.2.4.6** `lpuart_status_t LPUART_HAL_SetBaudRate ( LPUART_Type * base, 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>base</i>             | LPUART base pointer.             |
| <i>sourceClockIn-Hz</i> | LPUART source input clock in Hz. |
| <i>desiredBaud-Rate</i> | LPUART desired baud rate.        |

Returns

An error code or kStatus\_Success

**26.2.4.7** `static void LPUART_HAL_SetBaudRateDivisor ( LPUART_Type * base, uint32_t baudRateDivisor ) [inline], [static]`

Parameters

|                         |                                     |
|-------------------------|-------------------------------------|
| <i>base</i>             | LPUART base pointer.                |
| <i>baudRate-Divisor</i> | The baud rate modulo division "SBR" |

**26.2.4.8** `void LPUART_HAL_SetBitCountPerChar ( LPUART_Type * base, lpuart_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.

## LPUART HAL driver

### Parameters

|                         |                                                                         |
|-------------------------|-------------------------------------------------------------------------|
| <i>base</i>             | LPUART base pointer.                                                    |
| <i>bitCountPer-Char</i> | Number of bits per char (8, 9, or 10, depending on the LPUART instance) |

### 26.2.4.9 void LPUART\_HAL\_SetParityMode ( LPUART\_Type \* *base*, 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>base</i>            | LPUART base pointer.                                                 |
| <i>parityMode-Type</i> | Parity mode (enabled, disable, odd, even - see parity_mode_t struct) |

### 26.2.4.10 static void LPUART\_HAL\_SetStopBitCount ( LPUART\_Type \* *base*, 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>base</i>         | LPUART base pointer.                                              |
| <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

### 26.2.4.11 static uint32\_t LPUART\_HAL\_GetDataRegAddr ( LPUART\_Type \* *base* ) [inline], [static]

### Returns

LPUART tx/rx data register address.

26.2.4.12 void LPUART\_HAL\_SetIntMode ( LPUART\_Type \* *base*, lpuart\_interrupt\_t *interrupt*, bool *enable* )

## LPUART HAL driver

### Parameters

|                  |                                      |
|------------------|--------------------------------------|
| <i>base</i>      | LPUART module base pointer.          |
| <i>interrupt</i> | LPUART interrupt configuration data. |
| <i>enable</i>    | true: enable, false: disable.        |

### 26.2.4.13 `bool LPUART_HAL_GetIntMode ( LPUART_Type * base, lpuart_interrupt_t interrupt )`

### Parameters

|                  |                                      |
|------------------|--------------------------------------|
| <i>base</i>      | LPUART module base pointer.          |
| <i>interrupt</i> | LPUART interrupt configuration data. |

### Returns

true: enable, false: disable.

### 26.2.4.14 `static void LPUART_HAL_Putchar ( LPUART_Type * base, uint8_t data )` `[inline], [static]`

### Parameters

|             |                      |
|-------------|----------------------|
| <i>base</i> | LPUART Instance      |
| <i>data</i> | data to send (8-bit) |

### 26.2.4.15 `void LPUART_HAL_Putchar9 ( LPUART_Type * base, uint16_t data )`

### Parameters

|             |                      |
|-------------|----------------------|
| <i>base</i> | LPUART Instance      |
| <i>data</i> | data to send (9-bit) |

### 26.2.4.16 `lpuart_status_t LPUART_HAL_Putchar10 ( LPUART_Type * base, uint16_t data )`

Parameters

|             |                       |
|-------------|-----------------------|
| <i>base</i> | LPUART Instance       |
| <i>data</i> | data to send (10-bit) |

Returns

An error code or kStatus\_Success

**26.2.4.17 static void LPUART\_HAL\_Getchar ( LPUART\_Type \* *base*, uint8\_t \* *readData* )  
[inline], [static]**

Parameters

|                 |                                |
|-----------------|--------------------------------|
| <i>base</i>     | LPUART base pointer            |
| <i>readData</i> | Data read from receive (8-bit) |

**26.2.4.18 void LPUART\_HAL\_Getchar9 ( LPUART\_Type \* *base*, uint16\_t \* *readData* )**

Parameters

|                 |                                |
|-----------------|--------------------------------|
| <i>base</i>     | LPUART base pointer            |
| <i>readData</i> | Data read from receive (9-bit) |

**26.2.4.19 void LPUART\_HAL\_Getchar10 ( LPUART\_Type \* *base*, uint16\_t \* *readData* )**

Parameters

|                 |                                 |
|-----------------|---------------------------------|
| <i>base</i>     | LPUART base pointer             |
| <i>readData</i> | Data read from receive (10-bit) |

**26.2.4.20 void LPUART\_HAL\_SendDataPolling ( LPUART\_Type \* *base*, const uint8\_t \* *txBuff*, uint32\_t *txSize* )**

This function only supports 8-bit transaction.

## LPUART HAL driver

### Parameters

|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>base</i>   | LPUART module base pointer.                         |
| <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.            |

### 26.2.4.21 `lpuart_status_t LPUART_HAL_ReceiveDataPolling ( LPUART_Type * base, uint8_t * rxBuff, uint32_t rxSize )`

This function only supports 8-bit transaction.

### Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | LPUART module base pointer.                             |
| <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.

### 26.2.4.22 `bool LPUART_HAL_GetStatusFlag ( LPUART_Type * base, lpuart_status_flag_t statusFlag )`

### Parameters

|                   |                          |
|-------------------|--------------------------|
| <i>base</i>       | LPUART base pointer      |
| <i>statusFlag</i> | The status flag to query |

### Returns

Whether the current status flag is set(true) or not(false).

### 26.2.4.23 `lpuart_status_t LPUART_HAL_ClearStatusFlag ( LPUART_Type * base, lpuart_status_flag_t statusFlag )`

Parameters

|                   |                                     |
|-------------------|-------------------------------------|
| <i>base</i>       | LPUART base pointer                 |
| <i>statusFlag</i> | Desired LPUART status flag to clear |

Returns

An error code or kStatus\_Success

**26.2.4.24 static void LPUART\_HAL\_SetIdleChar ( LPUART\_Type \* *base*,  
lpuart\_idle\_char\_t *idleConfig* ) [inline], [static]**

Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>base</i>       | LPUART base pointer           |
| <i>idleConfig</i> | Idle characters configuration |

**26.2.4.25 static lpuart\_idle\_char\_t LPUART\_HAL\_GetIdleChar ( LPUART\_Type \* *base* )  
[inline], [static]**

Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | LPUART base pointer |
|-------------|---------------------|

Returns

idle characters configuration

**26.2.4.26 static bool LPUART\_HAL\_IsCurrentDataWithNoise ( LPUART\_Type \* *base* )  
[inline], [static]**

Parameters

|             |                      |
|-------------|----------------------|
| <i>base</i> | LPUART base pointer. |
|-------------|----------------------|

Returns

The status of the NOISY bit in the LPUART extended data register

**26.2.4.27 static bool LPUART\_HAL\_IsCurrentDataWithFrameError ( LPUART\_Type \* *base*  
) [inline], [static]**

## LPUART HAL driver

### Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | LPUART base pointer |
|-------------|---------------------|

### Returns

The status of the FRETSC bit in the LPUART extended data register

**26.2.4.28** `static void LPUART_HAL_SetTxSpecialChar ( LPUART_Type * base, uint8_t specialChar ) [inline], [static]`

### Parameters

|                    |                                                                                                                       |
|--------------------|-----------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | LPUART base pointer                                                                                                   |
| <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. |

**26.2.4.29** `static bool LPUART_HAL_IsCurrentDataWithParityError ( LPUART_Type * base ) [inline], [static]`

### Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | LPUART base pointer |
|-------------|---------------------|

### Returns

The status of the PARITYE bit in the LPUART extended data register

**26.2.4.30** `static bool LPUART_HAL_IsReceiveBufferEmpty ( LPUART_Type * base ) [inline], [static]`

### Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | LPUART base pointer |
|-------------|---------------------|

### Returns

TRUE if the receive-buffer is empty, else FALSE.

**26.2.4.31** `static bool LPUART_HAL_WasPreviousReceiverStateIdle ( LPUART_Type * base ) [inline], [static]`



Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | LPUART base pointer |
|-------------|---------------------|

Returns

TRUE if the previous BUS state was IDLE, else FALSE.

**26.2.4.32 static void LPUART\_HAL\_SetWaitModeOperation ( LPUART\_Type \* *base*, 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>base</i> | LPUART base pointer                                                     |
| <i>mode</i> | LPUART wait mode operation - operates or stops to operate in wait mode. |

**26.2.4.33 static lpuart\_operation\_config\_t LPUART\_HAL\_GetWaitModeOperation ( LPUART\_Type \* *base* ) [inline], [static]**

Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | LPUART base pointer |
|-------------|---------------------|

Returns

- LPUART wait mode operation configuration
- kLpuartOperates or kLpuartStops in wait mode

**26.2.4.34 void LPUART\_HAL\_SetLoopbackCmd ( LPUART\_Type \* *base*, 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.

## LPUART HAL driver

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | LPUART base pointer                                |
| <i>enable</i> | LPUART loopback mode - disabled (0) or enabled (1) |

### 26.2.4.35 void LPUART\_HAL\_SetSingleWireCmd ( LPUART\_Type \* *base*, 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>base</i>   | LPUART base pointer                                |
| <i>enable</i> | LPUART loopback mode - disabled (0) or enabled (1) |

### 26.2.4.36 static void LPUART\_HAL\_SetTxdirInSinglewireMode ( LPUART\_Type \* *base*, lpuart\_singlewire\_txdir\_t *direction* ) [inline], [static]

Parameters

|                  |                                                         |
|------------------|---------------------------------------------------------|
| <i>base</i>      | LPUART base pointer                                     |
| <i>direction</i> | LPUART single-wire transmit direction - input or output |

### 26.2.4.37 lpuart\_status\_t LPUART\_HAL\_SetReceiverInStandbyMode ( LPUART\_Type \* *base* )

Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | LPUART base pointer |
|-------------|---------------------|

Returns

Error code or kStatus\_Success

### 26.2.4.38 static void LPUART\_HAL\_PutReceiverInNormalMode ( LPUART\_Type \* *base* ) [inline], [static]

Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | LPUART base pointer |
|-------------|---------------------|

**26.2.4.39 static bool LPUART\_HAL\_IsReceiverInStandby ( LPUART\_Type \* *base* )  
[inline], [static]**

Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | LPUART base pointer |
|-------------|---------------------|

Returns

LPUART in normal more (0) or standby (1)

**26.2.4.40 static void LPUART\_HAL\_SetReceiverWakeupMode ( LPUART\_Type \* *base*,  
lpuart\_wakeup\_method\_t *method* ) [inline], [static]**

Parameters

|               |                                                                        |
|---------------|------------------------------------------------------------------------|
| <i>base</i>   | LPUART base pointer                                                    |
| <i>method</i> | LPUART wakeup method: 0 - Idle-line wake (default), 1 - addr-mark wake |

**26.2.4.41 static lpuart\_wakeup\_method\_t LPUART\_HAL\_GetReceiverWakeupMode ( LPUART\_Type \* *base* ) [inline], [static]**

Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | LPUART base pointer |
|-------------|---------------------|

Returns

LPUART wakeup method: kLpuartIdleLineWake: 0 - Idle-line wake (default), kLpuartAddrMarkWake: 1 - addr-mark wake

**26.2.4.42 void LPUART\_HAL\_SetIdleLineDetect ( LPUART\_Type \* *base*, 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.

## LPUART HAL driver

### Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>base</i>   | LPUART base pointer                                      |
| <i>config</i> | LPUART configuration data for idle line detect operation |

#### 26.2.4.43 **static void LPUART\_HAL\_SetBreakCharTransmitLength ( LPUART\_Type \* *base*, 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>base</i>   | LPUART base pointer                                                                                 |
| <i>length</i> | LPUART break character length setting: 0 - minimum 10-bit times (default), 1 - minimum 13-bit times |

#### 26.2.4.44 **static void LPUART\_HAL\_SetBreakCharDetectLength ( LPUART\_Type \* *base*, lpuart\_break\_char\_length\_t *length* ) [inline], [static]**

### Parameters

|               |                                                                                                     |
|---------------|-----------------------------------------------------------------------------------------------------|
| <i>base</i>   | LPUART base pointer                                                                                 |
| <i>length</i> | LPUART break character length setting: 0 - minimum 10-bit times (default), 1 - minimum 13-bit times |

#### 26.2.4.45 **static void LPUART\_HAL\_QueueBreakCharToSend ( LPUART\_Type \* *base*, bool *enable* ) [inline], [static]**

### Parameters

|               |                                                                                                      |
|---------------|------------------------------------------------------------------------------------------------------|
| <i>base</i>   | LPUART base pointer                                                                                  |
| <i>enable</i> | LPUART normal/queue break char - disabled (normal mode, default: 0) or enabled (queue break char: 1) |

#### 26.2.4.46 **static void LPUART\_HAL\_SetMatchAddressMode ( LPUART\_Type \* *base*, lpuart\_match\_config\_t *config* ) [inline], [static]**

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | LPUART base pointer                                |
| <i>config</i> | MATCFG: Configures the match addressing mode used. |

**26.2.4.47 void LPUART\_HAL\_SetMatchAddressReg1 ( LPUART\_Type \* *base*, 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>base</i>   | LPUART base pointer                                          |
| <i>enable</i> | Match address model enable (true)/disable (false)            |
| <i>value</i>  | Match address value to program into match address register 1 |

**26.2.4.48 void LPUART\_HAL\_SetMatchAddressReg2 ( LPUART\_Type \* *base*, 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>base</i>   | LPUART base pointer                                          |
| <i>enable</i> | Match address model enable (true)/disable (false)            |
| <i>value</i>  | Match address value to program into match address register 2 |

**26.2.4.49 static void LPUART\_HAL\_SetSendMsbFirstCmd ( LPUART\_Type \* *base*, 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

---

## LPUART HAL driver

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | LPUART base pointer                                   |
| <i>enable</i> | false - LSB (default, disabled), true - MSB (enabled) |

**26.2.4.50** `static void LPUART_HAL_SetReceiveResyncCmd ( LPUART_Type * base, bool enable ) [inline], [static]`

Parameters

|               |                                                                                                                         |
|---------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | LPUART base pointer                                                                                                     |
| <i>enable</i> | re-sync of received data word configuration: true - re-sync of received data word (default) false - disable the re-sync |

## 26.3 LPUART Peripheral driver

### 26.3.1 Overview

This section describes the programming interface of the LPUART Peripheral driver.

#### Data Structures

- struct [lpuart\\_dma\\_state\\_t](#)  
*Runtime state structure for UART driver with DMA. [More...](#)*
- struct [lpuart\\_dma\\_user\\_config\\_t](#)  
*LPUART configuration structure. [More...](#)*
- 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.*
- typedef void(\* [lpuart\\_tx\\_callback\\_t](#))(uint32\_t instance, void \*lpuartState)  
*UART transmit callback function type.*

#### Variables

- LPUART\_Type \*const [g\\_lpuartBase](#) [LPUART\_INSTANCE\_COUNT]  
*Table of base addresses for LPUART instances.*
- LPUART\_Type \*const [g\\_lpuartBase](#) [LPUART\_INSTANCE\_COUNT]  
*Table of base addresses for LPUART instances.*
- const IRQn\_Type [g\\_lpuartRxTxIrqId](#) [LPUART\_INSTANCE\_COUNT]  
*Table to save LPUART IRQ enumeration numbers defined in the CMSIS header file.*
- LPUART\_Type \*const [g\\_lpuartBase](#) [LPUART\_INSTANCE\_COUNT]  
*Table of base addresses for LPUART instances.*

#### LPUART DMA Driver

- [lpuart\\_status\\_t](#) LPUART\_DRV\_DmaInit (uint32\_t instance, [lpuart\\_dma\\_state\\_t](#) \*lpuartDmaStatePtr, const [lpuart\\_dma\\_user\\_config\\_t](#) \*lpuartUserConfig)  
*Initializes an LPUART instance to work with DMA.*
- [lpuart\\_status\\_t](#) LPUART\_DRV\_DmaDeinit (uint32\_t instance)

## LPUART Peripheral driver

- Shuts down the LPUART.*

  - [lpuart\\_status\\_t LPUART\\_DRV\\_DmaSendDataBlocking](#) (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\\_DmaSendData](#) (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\\_DmaGetTransmitStatus](#) (uint32\_t instance, uint32\_t \*bytesRemaining)

*Returns whether the previous LPUART-DMA transmit has finished.*
- [lpuart\\_status\\_t LPUART\\_DRV\\_DmaAbortSendingData](#) (uint32\_t instance)

*Terminates a non-blocking LPUART-DMA transmission early.*
- [lpuart\\_status\\_t LPUART\\_DRV\\_DmaReceiveDataBlocking](#) (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\\_DmaReceiveData](#) (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\\_DmaGetReceiveStatus](#) (uint32\_t instance, uint32\_t \*bytesRemaining)

*Returns whether the previous LPUART-DMA receive is complete.*
- [lpuart\\_status\\_t LPUART\\_DRV\\_DmaAbortReceivingData](#) (uint32\_t instance)

*Terminates a non-blocking LPUART-DMA receive early.*

## 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.*
- [lpuart\\_status\\_t 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\\_tx\\_callback\\_t LPUART\\_DRV\\_InstallTxCallback](#) (uint32\_t instance, [lpuart\\_tx\\_callback\\_t](#) function, uint8\_t \*txBuff, void \*callbackParam)

*Installs callback function for the LPUART transmit.*
- [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\\_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)



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.*
- [lpuart\\_status\\_t 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.*

## 26.3.2 Data Structure Documentation

### 26.3.2.1 struct [lpuart\\_dma\\_state\\_t](#)

#### Data Fields

- volatile bool [isTxBusy](#)  
*True if there is an active transmit.*

## LPUART Peripheral driver

- 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.*
- [dma\\_channel\\_t dmaLpuartTx](#)  
*Structure definition for the DMA channel.*
- [dma\\_channel\\_t dmaLpuartRx](#)  
*Structure definition for the DMA channel.*

### 26.3.2.1.0.46 Field Documentation

**26.3.2.1.0.46.1** volatile bool `lpuart_dma_state_t::isTxBusy`

**26.3.2.1.0.46.2** volatile bool `lpuart_dma_state_t::isRxBusy`

**26.3.2.1.0.46.3** volatile bool `lpuart_dma_state_t::isTxBlocking`

**26.3.2.1.0.46.4** volatile bool `lpuart_dma_state_t::isRxBlocking`

**26.3.2.1.0.46.5** semaphore\_t `lpuart_dma_state_t::txIrqSync`

**26.3.2.1.0.46.6** semaphore\_t `lpuart_dma_state_t::rxIrqSync`

### 26.3.2.2 struct `lpuart_dma_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).*

### 26.3.2.2.0.47 Field Documentation

#### 26.3.2.2.0.47.1 lpuart\_bit\_count\_per\_char\_t lpuart\_dma\_user\_config\_t::bitCountPerChar

### 26.3.2.3 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_tx_callback_t txCallback`  
*Callback to invoke after transmitting byte.*
- `void * txCallbackParam`  
*Transmit callback parameter pointer.*

## LPUART Peripheral driver

### 26.3.2.3.0.48 Field Documentation

- 26.3.2.3.0.48.1 `const uint8_t* lpuart_state_t::txBuff`
- 26.3.2.3.0.48.2 `uint8_t* lpuart_state_t::rxBuff`
- 26.3.2.3.0.48.3 `volatile size_t lpuart_state_t::txSize`
- 26.3.2.3.0.48.4 `volatile size_t lpuart_state_t::rxSize`
- 26.3.2.3.0.48.5 `volatile bool lpuart_state_t::isTxBusy`
- 26.3.2.3.0.48.6 `volatile bool lpuart_state_t::isRxBusy`
- 26.3.2.3.0.48.7 `volatile bool lpuart_state_t::isTxBlocking`
- 26.3.2.3.0.48.8 `volatile bool lpuart_state_t::isRxBlocking`
- 26.3.2.3.0.48.9 `semaphore_t lpuart_state_t::txIrqSync`
- 26.3.2.3.0.48.10 `semaphore_t lpuart_state_t::rxIrqSync`
- 26.3.2.3.0.48.11 `lpuart_rx_callback_t lpuart_state_t::rxCallback`
- 26.3.2.3.0.48.12 `void* lpuart_state_t::rxCallbackParam`
- 26.3.2.3.0.48.13 `lpuart_tx_callback_t lpuart_state_t::txCallback`
- 26.3.2.3.0.48.14 `void* lpuart_state_t::txCallbackParam`

### 26.3.2.4 struct `lpuart_user_config_t`

#### Data Fields

- `clock_lpuart_src_t` [clockSource](#)  
*LPUART clock source.*
- `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).*

### 26.3.2.5 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.*

#### 26.3.2.5.0.49 Field Documentation

26.3.2.5.0.49.1 volatile bool [lpuart\\_edma\\_state\\_t::isTxBusy](#)

26.3.2.5.0.49.2 volatile bool [lpuart\\_edma\\_state\\_t::isRxBusy](#)

26.3.2.5.0.49.3 volatile bool [lpuart\\_edma\\_state\\_t::isTxBlocking](#)

26.3.2.5.0.49.4 volatile bool [lpuart\\_edma\\_state\\_t::isRxBlocking](#)

26.3.2.5.0.49.5 semaphore\_t [lpuart\\_edma\\_state\\_t::txIrqSync](#)

26.3.2.5.0.49.6 semaphore\_t [lpuart\\_edma\\_state\\_t::rxIrqSync](#)

#### 26.3.2.6 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

### 26.3.2.6.0.50 Field Documentation

#### 26.3.2.6.0.50.1 `lpuart_bit_count_per_char_t lpuart_edma_user_config_t::bitCountPerChar`

### 26.3.3 Typedef Documentation

#### 26.3.3.1 `typedef void(* lpuart_rx_callback_t)(uint32_t instance, void *lpuartState)`

### 26.3.4 Function Documentation

#### 26.3.4.1 `lpuart_status_t LPUART_DRV_DmaInit ( uint32_t instance, lpuart_dma_state_t * lpuartDmaStatePtr, const lpuart_dma_user_config_t * lpuartUserConfig )`

This function initializes the run-time state structure to keep track of the on-going transfers, un-gates 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_dma_state_t` and the `lpuart_user_config_t` parameters and how to call the `LPUART_DRV_DmaInit` function by passing in these parameters:

```
lpuart_user_config_t lpuartConfig;
lpuartConfig.baudRate = 9600;
lpuartConfig.bitCountPerChar = kLpuart8BitsPerChar;
lpuartConfig.parityMode = kLpuartParityDisabled;
lpuartConfig.stopBitCount = kLpuartOneStopBit;
lpuart_dma_state_t lpuartDmaState;
LPUART_DRV_DmaInit(instance, &lpuartDmaState, &lpuartConfig);
```

#### Parameters

|                          |                                                                                                                                                                                                                                                |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>          | The LPUART instance number.                                                                                                                                                                                                                    |
| <i>lpuartDmaStatePtr</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>lpuartUserConfig</i>  | The user configuration structure of type <code>lpuart_user_config_t</code> . 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`.

#### 26.3.4.2 `lpuart_status_t LPUART_DRV_DmaDeinit ( uint32_t instance )`

This function disables the LPUART-DMA trigger, the transmitter, and the receiver.

Parameters

|                 |                             |
|-----------------|-----------------------------|
| <i>instance</i> | The LPUART instance number. |
|-----------------|-----------------------------|

Returns

An error code or kStatus\_LPUART\_Success.

**26.3.4.3** `lpuart_status_t LPUART_DRV_DmaSendDataBlocking ( 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.

**26.3.4.4** `lpuart_status_t LPUART_DRV_DmaSendData ( 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.

**26.3.4.5** `lpuart_status_t LPUART_DRV_DmaGetTransmitStatus ( uint32_t instance, uint32_t * bytesRemaining )`

## LPUART Peripheral driver

### Parameters

|                        |                                                                                                            |
|------------------------|------------------------------------------------------------------------------------------------------------|
| <i>instance</i>        | The LPUART module base address.                                                                            |
| <i>bytes-Remaining</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. |

### 26.3.4.6 `lpuart_status_t LPUART_DRV_DmaAbortSendingData ( 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. |

### 26.3.4.7 `lpuart_status_t LPUART_DRV_DmaReceiveDataBlocking ( uint32_t instance, uint8_t * rxBuff, uint32_t rxSize, uint32_t timeout )`



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.

**26.3.4.8 lpuart\_status\_t LPUART\_DRV\_DmaReceiveData ( 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.

**26.3.4.9 lpuart\_status\_t LPUART\_DRV\_DmaGetReceiveStatus ( 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.

## LPUART Peripheral driver

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. |

### 26.3.4.10 `lpuart_status_t LPUART_DRV_DmaAbortReceivingData ( 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. |

### 26.3.4.11 `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 <a href="#">lpuart_user_config_t</a> |

|                       |                                              |
|-----------------------|----------------------------------------------|
| <i>lpuartStatePtr</i> | pointer to the LPUART driver state structure |
|-----------------------|----------------------------------------------|

Returns

An error code or kStatus\_LPUART\_Success

#### 26.3.4.12 **lpuart\_status\_t LPUART\_DRV\_Deinit ( uint32\_t instance )**

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | LPUART instance number |
|-----------------|------------------------|

Returns

An error code or kStatus\_LPUART\_Success

#### 26.3.4.13 **lpuart\_rx\_callback\_t LPUART\_DRV\_InstallRxCallback ( uint32\_t instance, lpuart\_rx\_callback\_t function, uint8\_t \* rxBuff, void \* callbackParam, bool alwaysEnableRxIrq )**

Note

After a callback is installed, it bypasses part of the LPUART IRQHandler logic. Therefore, the callback needs to handle the indexes of txBuff and txSize.

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>alwaysEnable-RxIrq</i> | Whether always enable receive IRQ or not.                                                             |

Returns

Former LPUART receive callback function pointer.

## LPUART Peripheral driver

### 26.3.4.14 `lpuart_tx_callback_t LPUART_DRV_InstallTxCallback ( uint32_t instance, lpuart_tx_callback_t function, uint8_t * txBuff, void * callbackParam )`

Note

After a callback is installed, it bypasses part of the LPUART IRQHandler logic. Therefore, the callback needs to handle the indexes of txBuff and txSize.

Parameters

|                      |                                                                                                        |
|----------------------|--------------------------------------------------------------------------------------------------------|
| <i>instance</i>      | The LPUART instance number.                                                                            |
| <i>function</i>      | The LPUART transmit callback function.                                                                 |
| <i>txBuff</i>        | The transmit buffer used inside IRQHandler. This buffer must be kept as long as the callback is alive. |
| <i>callbackParam</i> | The LPUART transmit callback parameter pointer.                                                        |

Returns

Former LPUART transmit callback function pointer.

### 26.3.4.15 `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.

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`

### 26.3.4.16 `lpuart_status_t LPUART_DRV_SendData ( uint32_t instance, const uint8_t * txBuff, uint32_t txSize )`

This enables an a-sync 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`

**26.3.4.17** `lpuart_status_t LPUART_DRV_GetTransmitStatus ( uint32_t instance, uint32_t * bytesRemaining )`

Parameters

|                        |                                                                                                        |
|------------------------|--------------------------------------------------------------------------------------------------------|
| <i>instance</i>        | LPUART instance number                                                                                 |
| <i>bytes-Remaining</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. |

**26.3.4.18** `lpuart_status_t LPUART_DRV_AbortSendingData ( uint32_t instance )`

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | LPUART instance number |
|-----------------|------------------------|

Returns

Whether the aborting is successful or not.

## LPUART Peripheral driver

**26.3.4.19** `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

**26.3.4.20 lpuart\_status\_t LPUART\_DRV\_ReceiveData ( uint32\_t *instance*, uint8\_t \* *rxBuff*, uint32\_t *rxSize* )**

This enables an a-sync 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

**26.3.4.21 lpuart\_status\_t LPUART\_DRV\_GetReceiveStatus ( uint32\_t *instance*, uint32\_t \* *bytesRemaining* )**

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | LPUART instance number |
|-----------------|------------------------|

## LPUART Peripheral driver

|                        |                                                                                                                 |
|------------------------|-----------------------------------------------------------------------------------------------------------------|
| <i>bytes-Remaining</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_RxBusy</i>  | the receive is still in progress. <i>bytesReceived</i> will be filled with the number of bytes that have been received so far. |

### 26.3.4.22 `lpuart_status_t LPUART_DRV_AbortReceivingData ( uint32_t instance )`

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | LPUART instance number |
|-----------------|------------------------|

Returns

Whether the receiving was successful or not.

### 26.3.4.23 `lpuart_status_t LPUART_DRV_EdmaInit ( 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, un-gates 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_EdmaInit` 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_EdmaInit(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 <code>lpuart_user_config_t</code> . 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`.

**26.3.4.24 `lpuart_status_t LPUART_DRV_EdmaDeinit ( uint32_t instance )`**

This function disables the LPUART-DMA trigger, the transmitter, and the receiver.

Parameters

|                 |                             |
|-----------------|-----------------------------|
| <i>instance</i> | The LPUART instance number. |
|-----------------|-----------------------------|

Returns

An error code or `kStatus_LPUART_Success`.

**26.3.4.25 `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`.

**26.3.4.26 `lpuart_status_t LPUART_DRV_EdmaSendData ( uint32_t instance, const uint8_t * txBuff, uint32_t txSize )`**

## LPUART Peripheral driver

### 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`.

### 26.3.4.27 `lpuart_status_t` LPUART\_DRV\_EdmaGetTransmitStatus ( `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 is populated with the number of bytes that are remaining in the active transfer. |

### Returns

Current transmit status.

### Return values

|                                     |                                                                                                                                       |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <code>kStatus_LPUART_Success</code> | The transmit has completed successfully.                                                                                              |
| <code>kStatus_LPUART_TxBusy</code>  | The transmit is still in progress. <i>bytesTransmitted</i> is filled with the number of bytes which are transmitted up to that point. |

### 26.3.4.28 `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_NoTransmitInProgress</i> | No transmission is currently in progress. |

**26.3.4.29** `lpuart_status_t LPUART_DRV_EdmaReceiveDataBlocking ( uint32_t instance, uint8_t * rxBuff, uint32_t rxSize, uint32_t timeout )`

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`.

**26.3.4.30** `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`.

**26.3.4.31** `lpuart_status_t LPUART_DRV_EdmaGetReceiveStatus ( uint32_t instance, uint32_t * bytesRemaining )`

## LPUART Peripheral driver

### 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_Rx-Busy</i>  | The receive is still in progress. <i>bytesReceived</i> is filled with the number of bytes which are received up to that point. |

### 26.3.4.32 `lpuart_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_No-TransmitInProgress</i> | No receive is currently in progress. |

## 26.3.5 Variable Documentation

**26.3.5.1** `LPUART_Type* const g_lpuartBase[LPUART_INSTANCE_COUNT]`

**26.3.5.2** `LPUART_Type* const g_lpuartBase[LPUART_INSTANCE_COUNT]`

**26.3.5.3** `LPUART_Type* const g_lpuartBase[LPUART_INSTANCE_COUNT]`

## 26.4 LPUART Type Definitions

This section describes the LPUART type definitions.

### 26.4.1 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 call.

### 26.4.2 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.

### 26.4.3 LPUART Initialization

1. To initialize the LPUART driver, call the `LPUART_DRV_Init()` function and pass the instance number of the relevant LPUART peripheral. 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;
```

### 26.4.4 LPUART Transfers

The driver implements transmit and receive functions to transfer buffers of data by blocking and non-blocking modes.

---

## LPUART Type Definitions

The blocking transmit and receive functions include `LPUART_DRV_SendDataBlocking()` and the `LPUART_DRV_ReceiveDataBlocking()` functions.

The non-blocking (async) transmit and receive functions include the `LPUART_DRV_SendData()` and the `LPUART_DRV_ReceiveData()` functions.

In all these cases, the functions are interrupt-driven.

## Chapter 27

# Memory Protection Unit (MPU)

### 27.1 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](#)

## MPU HAL driver

### 27.2 MPU HAL driver

#### 27.2.1 Overview

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

#### Data Structures

- struct [mpu\\_hardware\\_info\\_t](#)  
*MPU hardware basic information. [More...](#)*
- struct [mpu\\_access\\_err\\_info\\_t](#)  
*Describes MPU detail error access info. [More...](#)*
- struct [mpu\\_low\\_masters\\_access\\_rights\\_t](#)  
*MPU access rights for low master0~master3. [More...](#)*
- struct [mpu\\_high\\_masters\\_access\\_rights\\_t](#)  
*MPU access rights mode for high master4~master7. [More...](#)*
- struct [mpu\\_region\\_config\\_t](#)  
*Data v for MPU region initialize. [More...](#)*

#### Enumerations

- enum [mpu\\_region\\_num\\_t](#)  
*MPU region number region0~region11.*
- enum [mpu\\_region\\_total\\_num\\_t](#) {  
[kMPU8Regions](#) = 0x0U,  
[kMPU12Regions](#) = 0x1U,  
[kMPU16Regions](#) = 0x2U }  
*Describes the number of MPU regions.*
- enum [mpu\\_err\\_access\\_type\\_t](#) {  
[kMPUErrTypeRead](#) = 0U,  
[kMPUErrTypeWrite](#) = 1U }  
*MPU access error.*
- enum [mpu\\_err\\_attributes\\_t](#) {  
[kMPUInstructionAccessInUserMode](#) = 0U,  
[kMPUDataAccessInUserMode](#) = 1U,  
[kMPUInstructionAccessInSupervisorMode](#) = 2U,  
[kMPUDataAccessInSupervisorMode](#) = 3U }  
*MPU access error attributes.*
- enum [mpu\\_access\\_mode\\_t](#) {  
[kMPUAccessInUserMode](#) = 0U,  
[kMPUAccessInSupervisorMode](#) = 1U }  
*access MPU in which mode.*
- enum [mpu\\_master\\_t](#) {



```

kMPUMaster0 = 0U,
kMPUMaster1 = 1U,
kMPUMaster2 = 2U,
kMPUMaster3 = 3U,
kMPUMaster4 = 4U,
kMPUMaster5 = 5U,
kMPUMaster6 = 6U }

```

*MPU master number.*

- enum `mpu_err_access_ctr_t` {
 

```

kMPUNoRegionHit = 0U,
kMPUNoneOverlappRegion = 1U,
kMPUOverlappRegion = 2U }

```
- enum `mpu_supervisor_access_rights_t` {
 

```

kMPUSupervisorReadWriteExecute = 0U,
kMPUSupervisorReadExecute = 1U,
kMPUSupervisorReadWrite = 2U,
kMPUSupervisorEqualToUsermode = 3U }

```

*MPU access rights in supervisor mode for master0~master3.*

- enum `mpu_user_access_rights_t` {
 

```

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_status_t` {
 

```

kStatus_MPU_Success = 0x0U,
kStatus_MPU_Fail = 0x1U,
kStatus_MPU_NotInitialized = 0x2U,
kStatus_MPU_NullArgument = 0x3U }

```

*MPU status return codes.*

## MPU HAL.

- static void `MPU_HAL_Enable` (MPU\_Type \*base)
 

*Enables the MPU module operation.*
- static void `MPU_HAL_Disable` (MPU\_Type \*base)
 

*Disables the MPU module operation.*
- static bool `MPU_HAL_IsEnable` (MPU\_Type \*base)
 

*Checks whether the MPU module is enabled.*
- void `MPU_HAL_GetHardwareInfo` (MPU\_Type \*base, `mpu_hardware_info_t` \*infoPtr)
 

*Gets MPU basic hardware info.*

## MPU HAL driver

- void [MPU\\_HAL\\_GetDetailErrorAccessInfo](#) (MPU\_Type \*base, [mpu\\_access\\_err\\_info\\_t](#) \*errInfoArrayPtr)  
*Gets MPU derail error access info.*
- void [MPU\\_HAL\\_SetRegionAddr](#) (MPU\_Type \*base, [mpu\\_region\\_num\\_t](#) regionNum, uint32\_t startAddr, uint32\_t endAddr)  
*Sets region start and end address.*
- void [MPU\\_HAL\\_SetLowMasterAccessRights](#) (MPU\_Type \*base, [mpu\\_region\\_num\\_t](#) regionNum, [mpu\\_master\\_t](#) masterNum, const [mpu\\_low\\_masters\\_access\\_rights\\_t](#) \*accessRightsPtr)  
*Configures low master0~3 access permission for a specific region.*
- void [MPU\\_HAL\\_SetHighMasterAccessRights](#) (MPU\_Type \*base, [mpu\\_region\\_num\\_t](#) regionNum, [mpu\\_master\\_t](#) masterNum, const [mpu\\_high\\_masters\\_access\\_rights\\_t](#) \*accessRightsPtr)  
*Sets high master access permission for a specific region.*
- static void [MPU\\_HAL\\_SetRegionValidCmd](#) (MPU\_Type \*base, [mpu\\_region\\_num\\_t](#) regionNum, bool enable)  
*Sets the region valid value.*
- void [MPU\\_HAL\\_SetLowMasterAccessRightsByAlternateReg](#) (MPU\_Type \*base, [mpu\\_region\\_num\\_t](#) regionNum, [mpu\\_master\\_t](#) masterNum, const [mpu\\_low\\_masters\\_access\\_rights\\_t](#) \*accessRightsPtr)  
*Configures low master0~3 access permission for a specific region.*
- void [MPU\\_HAL\\_SetHighMasterAccessRightsByAlternateReg](#) (MPU\_Type \*base, [mpu\\_region\\_num\\_t](#) regionNum, [mpu\\_master\\_t](#) masterNum, const [mpu\\_high\\_masters\\_access\\_rights\\_t](#) \*accessRightsPtr)  
*Sets high master access permission for a specific region.*
- void [MPU\\_HAL\\_SetRegionConfig](#) (MPU\_Type \*base, const [mpu\\_region\\_config\\_t](#) \*regionConfigPtr)  
*Configures the MPU region.*
- void [MPU\\_HAL\\_Init](#) (MPU\_Type \*base)  
*Initializes the MPU module.*

## 27.2.2 Data Structure Documentation

### 27.2.2.1 struct [mpu\\_hardware\\_info\\_t](#)

#### Data Fields

- [uint8\\_t kMPUHardwareRevisionLevel](#)  
*Specifies the MPU's hardware and definition reversion level.*
- [uint8\\_t kMPUSupportSlavePortsNum](#)  
*Specifies the number of slave ports connected to MPU.*
- [mpu\\_region\\_total\\_num\\_t kMPUSupportRegionsNum](#)  
*Indicates the number of region descriptors implemented.*

### 27.2.2.2 struct [mpu\\_access\\_err\\_info\\_t](#)

#### Data Fields

- [mpu\\_master\\_t](#) master

- *Access error master.*
- [mpu\\_err\\_attributes\\_t](#) `attributes`  
*Access error attributes.*
- [mpu\\_err\\_access\\_type\\_t](#) `accessType`  
*Access error type.*
- [mpu\\_err\\_access\\_ctr\\_t](#) `accessCtr`  
*Access error control.*
- [uint32\\_t](#) `addr`  
*Access error address.*
- [uint8\\_t](#) `slavePort`  
*Access error slave port.*

### 27.2.2.3 struct mpu\_low\_masters\_access\_rights\_t

#### Data Fields

- [mpu\\_supervisor\\_access\\_rights\\_t](#) `superAccessRights`  
*master access rights in supervisor mode*
- [mpu\\_user\\_access\\_rights\\_t](#) `userAccessRights`  
*master access rights in user mode*

### 27.2.2.4 struct mpu\_high\_masters\_access\_rights\_t

#### Data Fields

- [bool](#) `kMPUWriteEnable`  
*Enables or disables write permission.*
- [bool](#) `kMPUReadEnable`  
*Enables or disables read permission.*

### 27.2.2.5 struct mpu\_region\_config\_t

This structure is used when calling the `MPU_DRV_Init` function.

#### Data Fields

- [mpu\\_region\\_num\\_t](#) `regionNum`  
*MPU region number.*
- [uint32\\_t](#) `startAddr`  
*Memory region start address.*
- [uint32\\_t](#) `endAddr`  
*Memory region end address.*
- [mpu\\_low\\_masters\\_access\\_rights\\_t](#) `accessRights1` [4]  
*Low masters access permission.*
- [mpu\\_high\\_masters\\_access\\_rights\\_t](#) `accessRights2` [4]  
*Low masters access permission.*
- [bool](#) `regionEnable`

## MPU HAL driver

*Enables or disables region.*

### 27.2.3 Enumeration Type Documentation

#### 27.2.3.1 enum mpu\_region\_num\_t

#### 27.2.3.2 enum mpu\_region\_total\_num\_t

Enumerator

*kMPU8Regions* MPU supports 8 regions.

*kMPU12Regions* MPU supports 12 regions.

*kMPU16Regions* MPU supports 16 regions.

#### 27.2.3.3 enum mpu\_err\_access\_type\_t

Enumerator

*kMPUErrTypeRead* MPU error type—read.

*kMPUErrTypeWrite* MPU error type—write.

#### 27.2.3.4 enum mpu\_err\_attributes\_t

Enumerator

*kMPUInstructionAccessInUserMode* access instruction error in user mode

*kMPUDataAccessInUserMode* access data error in user mode

*kMPUInstructionAccessInSupervisorMode* access instruction error in supervisor mode

*kMPUDataAccessInSupervisorMode* access data error in supervisor mode

#### 27.2.3.5 enum mpu\_access\_mode\_t

Enumerator

*kMPUAccessInUserMode* access data or instruction in user mode

*kMPUAccessInSupervisorMode* access data or instruction in supervisor mode

#### 27.2.3.6 enum mpu\_master\_t

Enumerator

*kMPUMaster0* MPU master core.

*kMPUMaster1* MPU master defined in SOC.  
*kMPUMaster2* MPU master defined in SOC.  
*kMPUMaster3* MPU master defined in SOC.  
*kMPUMaster4* MPU master defined in SOC.  
*kMPUMaster5* MPU master defined in SOC.  
*kMPUMaster6* MPU master defined in SOC.

### 27.2.3.7 enum mpu\_err\_access\_ctr\_t

Enumerator

*kMPUNoRegionHit* no region hit error  
*kMPUNoneOverlappRegion* access single region error  
*kMPUOverlappRegion* access overlapping region error

### 27.2.3.8 enum mpu\_supervisor\_access\_rights\_t

Enumerator

*kMPUSupervisorReadWriteExecute* Read write and execute operations are allowed in supervisor mode.  
*kMPUSupervisorReadExecute* Read and execute operations are allowed in supervisor mode.  
*kMPUSupervisorReadWrite* Read write operations are allowed in supervisor mode.  
*kMPUSupervisorEqualToUsermode* Access permission equal to user mode.

### 27.2.3.9 enum mpu\_user\_access\_rights\_t

Enumerator

*kMPUUserNoAccessRights* no access allowed in user mode  
*kMPUUserExecute* execute operation is allowed in user mode  
*kMPUUserWrite* Write operation is allowed in user mode.  
*kMPUUserWriteExecute* Write and execute operations are allowed in user mode.  
*kMPUUserRead* Read is allowed in user mode.  
*kMPUUserReadExecute* Read and execute operations are allowed in user mode.  
*kMPUUserReadWrite* Read and write operations are allowed in user mode.  
*kMPUUserReadWriteExecute* Read write and execute operations are allowed in user mode.

### 27.2.3.10 enum mpu\_status\_t

Enumerator

*kStatus\_MPU\_Success* MPU Succeed.

## MPU HAL driver

*kStatus\_MPU\_Fail* MPU failed.

*kStatus\_MPU\_NotInitialized* MPU is not initialized yet.

*kStatus\_MPU\_NullArgument* Argument is NULL.

### 27.2.4 Function Documentation

#### 27.2.4.1 static void MPU\_HAL\_Enable ( MPU\_Type \* *base* ) [inline], [static]

Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>base</i> | Base address of MPU peripheral instance. |
|-------------|------------------------------------------|

#### 27.2.4.2 static void MPU\_HAL\_Disable ( MPU\_Type \* *base* ) [inline], [static]

Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>base</i> | Base address of MPU peripheral instance. |
|-------------|------------------------------------------|

#### 27.2.4.3 static bool MPU\_HAL\_IsEnable ( MPU\_Type \* *base* ) [inline], [static]

Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>base</i> | Base address of MPU peripheral instance. |
|-------------|------------------------------------------|

Returns

State of the module

Return values

|              |                         |
|--------------|-------------------------|
| <i>true</i>  | MPU module is enabled.  |
| <i>false</i> | MPU module is disabled. |

#### 27.2.4.4 void MPU\_HAL\_GetHardwareInfo ( MPU\_Type \* *base*, mpu\_hardware\_info\_t \* *infoPtr* )

## Parameters

|                |                                                                                             |
|----------------|---------------------------------------------------------------------------------------------|
| <i>base</i>    | Base address of MPU peripheral instance.                                                    |
| <i>infoPtr</i> | The pointer to the hardware information structure see <a href="#">mpu_hardware_info_t</a> . |

**27.2.4.5 void MPU\_HAL\_GetDetailErrorAccessInfo ( MPU\_Type \* base, mpu\_access\_err\_info\_t \* errInfoArrayPtr )**

## Parameters

|                        |                                                                           |
|------------------------|---------------------------------------------------------------------------|
| <i>base</i>            | Base address of MPU peripheral instance.                                  |
| <i>errInfoArrayPtr</i> | The pointer to array of structure <a href="#">mpu_access_err_info_t</a> . |

**27.2.4.6 void MPU\_HAL\_SetRegionAddr ( MPU\_Type \* base, mpu\_region\_num\_t regionNum, uint32\_t startAddr, uint32\_t endAddr )**

## Parameters

|                  |                                           |
|------------------|-------------------------------------------|
| <i>base</i>      | Base address of MPU peripheral instance.. |
| <i>regionNum</i> | MPU region number.                        |
| <i>startAddr</i> | Region start address.                     |
| <i>endAddr</i>   | Region end address.                       |

**27.2.4.7 void MPU\_HAL\_SetLowMasterAccessRights ( MPU\_Type \* base, mpu\_region\_num\_t regionNum, mpu\_master\_t masterNum, const mpu\_low\_masters\_access\_rights\_t \* accessRightsPtr )**

## Parameters

|                  |                                          |
|------------------|------------------------------------------|
| <i>base</i>      | Base address of MPU peripheral instance. |
| <i>regionNum</i> | MPU region number.                       |

## MPU HAL driver

|                        |                                                                                           |
|------------------------|-------------------------------------------------------------------------------------------|
| <i>masterNum</i>       | MPU master number.                                                                        |
| <i>accessRightsPtr</i> | The pointer of master access rights see <a href="#">mpu_low_masters_access_rights_t</a> . |

**27.2.4.8 void MPU\_HAL\_SetHighMasterAccessRights ( MPU\_Type \* *base*, mpu\_region\_num\_t *regionNum*, mpu\_master\_t *masterNum*, const mpu\_high\_masters\_access\_rights\_t \* *accessRightsPtr* )**

Parameters

|                        |                                                                                           |
|------------------------|-------------------------------------------------------------------------------------------|
| <i>base</i>            | Base address of MPU peripheral instance.                                                  |
| <i>regionNum</i>       | MPU region number.                                                                        |
| <i>masterNum</i>       | MPU master number.                                                                        |
| <i>accessRightsPtr</i> | The pointer of master access rights see <a href="#">mpu_low_masters_access_rights_t</a> . |

**27.2.4.9 static void MPU\_HAL\_SetRegionValidCmd ( MPU\_Type \* *base*, mpu\_region\_num\_t *regionNum*, bool *enable* ) [inline], [static]**

When a region changed not by alternating registers should set the valid again.

Parameters

|                  |                                          |
|------------------|------------------------------------------|
| <i>base</i>      | Base address of MPU peripheral instance. |
| <i>regionNum</i> | MPU region number.                       |
| <i>enable</i>    | Enables or disables region.              |

**27.2.4.10 void MPU\_HAL\_SetLowMasterAccessRightsByAlternateReg ( MPU\_Type \* *base*, mpu\_region\_num\_t *regionNum*, mpu\_master\_t *masterNum*, const mpu\_low\_masters\_access\_rights\_t \* *accessRightsPtr* )**

Parameters



|                        |                                                                                           |
|------------------------|-------------------------------------------------------------------------------------------|
| <i>base</i>            | Base address of MPU peripheral instance.                                                  |
| <i>regionNum</i>       | MPU region number.                                                                        |
| <i>masterNum</i>       | MPU master number.                                                                        |
| <i>accessRightsPtr</i> | The pointer of master access rights see <a href="#">mpu_low_masters_access_rights_t</a> . |

**27.2.4.11 void MPU\_HAL\_SetHighMasterAccessRightsByAlternateReg ( MPU\_Type \* *base*, mpu\_region\_num\_t *regionNum*, mpu\_master\_t *masterNum*, const mpu\_high\_masters\_access\_rights\_t \* *accessRightsPtr* )**

Parameters

|                        |                                                                                           |
|------------------------|-------------------------------------------------------------------------------------------|
| <i>base</i>            | Base address of MPU peripheral instance.                                                  |
| <i>regionNum</i>       | MPU region number.                                                                        |
| <i>masterNum</i>       | MPU master number.                                                                        |
| <i>accessRightsPtr</i> | The pointer of master access rights see <a href="#">mpu_low_masters_access_rights_t</a> . |

**27.2.4.12 void MPU\_HAL\_SetRegionConfig ( MPU\_Type \* *base*, const mpu\_region\_config\_t \* *regionConfigPtr* )**

Parameters

|                        |                                                                                            |
|------------------------|--------------------------------------------------------------------------------------------|
| <i>base</i>            | The MPU peripheral base address.                                                           |
| <i>regionConfigPtr</i> | The pointer to the MPU user configure structure, see <a href="#">mpu_region_config_t</a> . |

**27.2.4.13 void MPU\_HAL\_Init ( MPU\_Type \* *base* )**

Parameters

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

## MPU Peripheral driver

### 27.3 MPU Peripheral driver

#### 27.3.1 Overview

This section describes the programming interface of the MPU Peripheral driver. The MPU driver configures MPU.

#### 27.3.2 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 = kMPUIdentifierDisable;
 .m1UserMode = kMPUUserNoAccessRights;
 .m1SupervisorMode = kMPUSupervisorEqualToUsermode;
 .m1Process_identifier = kMPUIdentifierDisable;
 .m2UserMode = kMPUUserNoAccessRights;
 .m2SupervisorMode = kMPUSupervisorEqualToUsermode;
 .m2Process_identifier = kMPUIdentifierDisable;
 .m3UserMode = kMPUUserNoAccessRights;
 .m3SupervisorMode = kMPUSupervisorEqualToUsermode;
 .m3Process_identifier = kMPUIdentifierDisable;
 .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);
```

### 27.3.3 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\\_user\\_config\\_t](#)  
Data The chapter describes the programming interface of the for MPU region initialization. [More...](#)

### Variables

- MPU\_Type \*const [g\\_mpuBase](#) []  
Table of base addresses for MPU instances.
- const IRQn\_Type [g\\_mpuIrqId](#) [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, const [mpu\\_user\\_config\\_t](#) \*userConfigPtr)  
*Initializes the MPU driver.*
- void [MPU\\_DRV\\_Deinit](#) (uint32\_t instance)  
*De-initializes the MPU region.*
- [mpu\\_status\\_t MPU\\_DRV\\_SetRegionConfig](#) (uint32\_t instance, const [mpu\\_region\\_config\\_t](#) \*regionConfigPtr)  
*Configures the MPU region.*
- void [MPU\\_DRV\\_SetRegionAddr](#) (uint32\_t instance, [mpu\\_region\\_num\\_t](#) regionNum, uint32\_t startAddr, uint32\_t endAddr)  
*Sets region start address.*
- [mpu\\_status\\_t MPU\\_DRV\\_SetLowMasterAccessRights](#) (uint32\_t instance, [mpu\\_region\\_num\\_t](#) regionNum, [mpu\\_master\\_t](#) masterNum, const [mpu\\_low\\_masters\\_access\\_rights\\_t](#) \*accessRightsPtr)

## MPU Peripheral driver

- Configures low master access permission.*

  - [mpu\\_status\\_t MPU\\_DRV\\_SetHighMasterAccessRights](#) (uint32\_t instance, [mpu\\_region\\_num\\_t](#) regionNum, [mpu\\_master\\_t](#) masterNum, const [mpu\\_high\\_masters\\_access\\_rights\\_t](#) \*accessRightsPtr)
- Configures high master access permission.*

  - void [MPU\\_DRV\\_SetRegionValidCmd](#) (uint32\_t instance, [mpu\\_region\\_num\\_t](#) regionNum, bool enable)
- Sets the MPU region valid.*

  - [mpu\\_status\\_t MPU\\_DRV\\_GetDetailErrorAccessInfo](#) (uint32\_t instance, [mpu\\_access\\_err\\_info\\_t](#) \*errInfoArrayPtr)

*Gets the MPU access error detail information.*

### 27.3.4 Data Structure Documentation

#### 27.3.4.1 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*

### 27.3.5 Function Documentation

#### 27.3.5.1 mpu\_status\_t MPU\_DRV\_Init ( uint32\_t instance, const mpu\_user\_config\_t \* userConfigPtr )

Parameters

|                      |                                                                                          |
|----------------------|------------------------------------------------------------------------------------------|
| <i>instance</i>      | The MPU peripheral instance number.                                                      |
| <i>userConfigPtr</i> | The pointer to the MPU user configure structure, see <a href="#">mpu_user_config_t</a> . |
| <i>userStatePtr</i>  | The pointer of run time structure.                                                       |

Returns

kStatus\_MPU\_Success means success. Otherwise, means failure.

#### 27.3.5.2 void MPU\_DRV\_Deinit ( uint32\_t instance )

## Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The MPU peripheral instance number. |
|-----------------|-------------------------------------|

## Returns

kStatus\_MPU\_Success means success. Otherwise, means failure.

**27.3.5.3 mpu\_status\_t MPU\_DRV\_SetRegionConfig ( uint32\_t *instance*, const mpu\_region\_config\_t \* *regionConfigPtr* )**

## Parameters

|                        |                                                                                            |
|------------------------|--------------------------------------------------------------------------------------------|
| <i>instance</i>        | The MPU peripheral instance number.                                                        |
| <i>regionConfigPtr</i> | The pointer to the MPU user configure structure, see <a href="#">mpu_region_config_t</a> . |

## Returns

kStatus\_MPU\_Success means success. Otherwise, means failure.

**27.3.5.4 void MPU\_DRV\_SetRegionAddr ( uint32\_t *instance*, mpu\_region\_num\_t *regionNum*, uint32\_t *startAddr*, uint32\_t *endAddr* )**

## Parameters

|                  |                                     |
|------------------|-------------------------------------|
| <i>instance</i>  | The MPU peripheral instance number. |
| <i>regionNum</i> | The region number.                  |
| <i>startAddr</i> | Region start address.               |
| <i>endAddr</i>   | Region end address.                 |

**27.3.5.5 mpu\_status\_t MPU\_DRV\_SetLowMasterAccessRights ( uint32\_t *instance*, mpu\_region\_num\_t *regionNum*, mpu\_master\_t *masterNum*, const mpu\_low\_masters\_access\_rights\_t \* *accessRightsPtr* )**

## MPU Peripheral driver

### Parameters

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>instance</i>        | The MPU peripheral instance number.       |
| <i>regionNum</i>       | The MPU region number.                    |
| <i>masterNum</i>       | The MPU master number.                    |
| <i>accessRightsPtr</i> | A pointer to access permission structure. |

### Returns

kStatus\_MPU\_Success means success. Otherwise, means failure.

### 27.3.5.6 `mpu_status_t MPU_DRV_SetHighMasterAccessRights ( uint32_t instance, mpu_region_num_t regionNum, mpu_master_t masterNum, const mpu_high_masters_access_rights_t * accessRightsPtr )`

### Parameters

|                        |                                           |
|------------------------|-------------------------------------------|
| <i>instance</i>        | The MPU peripheral instance number.       |
| <i>regionNum</i>       | The MPU region number.                    |
| <i>masterNum</i>       | The MPU master number.                    |
| <i>accessRightsPtr</i> | A pointer to access permission structure. |

### Returns

kStatus\_MPU\_Success means success. Otherwise, means failure.

### 27.3.5.7 `void MPU_DRV_SetRegionValidCmd ( uint32_t instance, mpu_region_num_t regionNum, bool enable )`

### Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The MPU peripheral instance number. |
|-----------------|-------------------------------------|

|                  |                             |
|------------------|-----------------------------|
| <i>regionNum</i> | MPU region number.          |
| <i>enable</i>    | Enables or disables region. |

**27.3.5.8 mpu\_status\_t MPU\_DRV\_GetDetailErrorAccessInfo ( uint32\_t instance, mpu\_access\_err\_info\_t \* errInfoArrayPtr )**

Parameters

|                         |                                           |
|-------------------------|-------------------------------------------|
| <i>instance</i>         | The MPU peripheral instance number.       |
| <i>errInfoArray-Ptr</i> | A pointer to access error info structure. |

**27.3.6 Variable Documentation**

**27.3.6.1 MPU\_Type\* const g\_mpuBase[]**

**27.3.6.2 const IRQn\_Type g\_mpuIrqId[MPU\_INSTANCE\_COUNT]**





## Chapter 28

# Programmable Delay Block (PDB)

### 28.1 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](#)

## PDB HAL driver

### 28.2 PDB HAL driver

#### 28.2.1 Overview

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

#### Data Structures

- struct `pdb_timer_config_t`  
*Defines the type of structure for basic timer in PDB. [More...](#)*

#### Enumerations

- enum `pdb_status_t` {  
`kStatus_PDB_Success` = 0U,  
`kStatus_PDB_InvalidArgument` = 1U,  
`kStatus_PDB_Failed` = 2U }  
*PDB status return codes.*
- enum `pdb_load_value_mode_t` {  
`kPdbLoadValueImmediately` = 0U,  
`kPdbLoadValueAtModuloCounter` = 1U,  
`kPdbLoadValueAtNextTrigger` = 2U,  
`kPdbLoadValueAtModuloCounterOrNextTrigger` = 3U }  
*Defines the type of value load mode for the PDB module.*
- enum `pdb_clk_prescaler_div_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_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_clk_prescaler_mult_factor_t` {
 

```

kPdbClkPreMultFactorAs1 = 0U,
kPdbClkPreMultFactorAs10 = 1U,
kPdbClkPreMultFactorAs20 = 2U,
kPdbClkPreMultFactorAs40 = 3U }

```

*Defines the type of the multiplication source mode for PDB.*

## Functions

- void `PDB_HAL_Init` (PDB\_Type \*base)
 

*Resets the PDB registers to a known state.*
- `pdb_status_t PDB_HAL_ConfigTimer` (PDB\_Type \*base, const `pdb_timer_config_t` \*configPtr)
 

*Configure the PDB timer.*
- static void `PDB_HAL_SetSoftTriggerCmd` (PDB\_Type \*base)
 

*Triggers the DAC by software if enabled.*
- static void `PDB_HAL_Enable` (PDB\_Type \*base)
 

*Switches on to enable the PDB module.*
- static void `PDB_HAL_Disable` (PDB\_Type \*base)
 

*Switches to disable the PDB module.*
- static bool `PDB_HAL_GetTimerIntFlag` (PDB\_Type \*base)
 

*Gets the PDB delay interrupt flag.*
- static void `PDB_HAL_ClearTimerIntFlag` (PDB\_Type \*base)
 

*Clears the PDB delay interrupt flag.*
- static void `PDB_HAL_SetLoadValuesCmd` (PDB\_Type \*base)
 

*Loads the delay registers value for the PDB module.*
- static void `PDB_HAL_SetTimerModulusValue` (PDB\_Type \*base, uint32\_t value)
 

*Sets the modulus value for the PDB module.*
- static uint32\_t `PDB_HAL_GetTimerValue` (PDB\_Type \*base)
 

*Gets the PDB counter value of PDB timer.*
- static void `PDB_HAL_SetValueForTimerInterrupt` (PDB\_Type \*base, uint32\_t value)

## PDB HAL driver

- Sets the interrupt delay milestone of the PDB counter.*

  - void [PDB\\_HAL\\_SetAdcPreTriggerBackToBackEnable](#) (PDB\_Type \*base, uint32\_t chn, uint32\_t preChnMask, bool enable)
- Switches to enable the pre-trigger back-to-back mode.*

  - void [PDB\\_HAL\\_SetAdcPreTriggerOutputEnable](#) (PDB\_Type \*base, uint32\_t chn, uint32\_t preChnMask, bool enable)
- Switches to enable the pre-trigger output.*

  - void [PDB\\_HAL\\_SetAdcPreTriggerEnable](#) (PDB\_Type \*base, uint32\_t chn, uint32\_t preChnMask, bool enable)
- Switches to enable the pre-trigger.*

  - static uint32\_t [PDB\\_HAL\\_GetAdcPreTriggerFlags](#) (PDB\_Type \*base, uint32\_t chn, uint32\_t preChnMask)
- Gets the flag which indicates whether the PDB counter has reached the pre-trigger delay value.*

  - void [PDB\\_HAL\\_ClearAdcPreTriggerFlags](#) (PDB\_Type \*base, uint32\_t chn, uint32\_t preChnMask)
- Clears the flag which indicates that the PDB counter has reached the pre-trigger delay value.*

  - static uint32\_t [PDB\\_HAL\\_GetAdcPreTriggerSeqErrFlags](#) (PDB\_Type \*base, uint32\_t chn, uint32\_t preChnMask)
- Gets the flag which indicates whether a sequence error is detected.*

  - void [PDB\\_HAL\\_ClearAdcPreTriggerSeqErrFlags](#) (PDB\_Type \*base, uint32\_t chn, uint32\_t preChnMask)
- Clears the flag which indicates that a sequence error has been detected.*

  - void [PDB\\_HAL\\_SetAdcPreTriggerDelayValue](#) (PDB\_Type \*base, uint32\_t chn, uint32\_t preChn, uint32\_t value)
- Sets the pre-trigger delay value.*

  - static void [PDB\\_HAL\\_SetDacExtTriggerInputEnable](#) (PDB\_Type \*base, uint32\_t dacChn, bool enable)
- Switches to enable the DAC external trigger input.*

  - static void [PDB\\_HAL\\_SetDacIntervalTriggerEnable](#) (PDB\_Type \*base, uint32\_t dacChn, bool enable)
- Switches to enable the DAC external trigger input.*

  - static void [PDB\\_HAL\\_SetDacIntervalValue](#) (PDB\_Type \*base, uint32\_t dacChn, uint32\_t value)
- Sets the interval value for the DAC trigger.*

  - void [PDB\\_HAL\\_SetCmpPulseOutEnable](#) (PDB\_Type \*base, uint32\_t pulseChnMask, bool enable)
- Switches to enable the pulse-out trigger.*

  - static void [PDB\\_HAL\\_SetCmpPulseOutDelayForHigh](#) (PDB\_Type \*base, uint32\_t pulseChn, uint32\_t value)
- Sets the counter delay value for the pulse-out goes high.*

  - static void [PDB\\_HAL\\_SetCmpPulseOutDelayForLow](#) (PDB\_Type \*base, uint32\_t pulseChn, uint32\_t value)
- Sets the counter delay value for the pulse-out goes low.*

## 28.2.2 Data Structure Documentation

### 28.2.2.1 struct pdb\_timer\_config\_t

#### Data Fields

- [pdb\\_load\\_value\\_mode\\_t loadValueMode](#)

- *Select the load mode.*
- bool `seqErrIntEnable`  
*Enable PDB Sequence Error Interrupt.*
- `pdb_clk_prescaler_div_t clkPreDiv`  
*Select the prescaler divider.*
- `pdb_clk_prescaler_mult_factor_t clkPreMultFactor`  
*Select multiplication factor for prescaler.*
- `pdb_trigger_src_t triggerInput`  
*Select the trigger input source.*
- bool `continuousModeEnable`  
*Enable the continuous mode.*
- bool `dmaEnable`  
*Enable the dma for timer.*
- bool `intEnable`  
*Enable the interrupt for timer.*

## 28.2.3 Enumeration Type Documentation

### 28.2.3.1 enum `pdb_status_t`

Enumerator

- kStatus\_PDB\_Success*** Success.
- kStatus\_PDB\_InvalidArgument*** Invalid argument existed.
- kStatus\_PDB\_Failed*** Execution failed.

### 28.2.3.2 enum `pdb_load_value_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

- kPdbLoadValueImmediately*** Loaded immediately after load operation.
- kPdbLoadValueAtModuloCounter*** Loaded when counter hits the modulo after load operation.
- kPdbLoadValueAtNextTrigger*** Loaded when detecting an input trigger after load operation.
- kPdbLoadValueAtModuloCounterOrNextTrigger*** Loaded when counter hits the modulo or detecting an input trigger after load operation.

### 28.2.3.3 enum `pdb_clk_prescaler_div_t`

Enumerator

- kPdbClkPreDivBy1*** Counting divided by multiplication factor selected by MULT.

## PDB HAL driver

- 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.

### 28.2.3.4 enum pdb\_trigger\_src\_t

Selects the trigger input source for the PDB. The trigger input source can be internal or external (EXTRG pin), or the software trigger.

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.

### 28.2.3.5 enum pdb\_clk\_prescaler\_mult\_factor\_t

Selects the multiplication factor of the prescaler divider for the PDB counter clock.

Enumerator

- kPdbClkPreMultFactorAs1* Multiplication factor is 1.
- kPdbClkPreMultFactorAs10* Multiplication factor is 10.
- kPdbClkPreMultFactorAs20* Multiplication factor is 20.
- kPdbClkPreMultFactorAs40* Multiplication factor is 40.

## 28.2.4 Function Documentation

### 28.2.4.1 void PDB\_HAL\_Init ( PDB\_Type \* *base* )

This function resets the PDB registers to a known state. This state is defined in a reference manual and is power on reset value.

## PDB HAL driver

Parameters

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

### 28.2.4.2 `pdb_status_t PDB_HAL_ConfigTimer ( PDB_Type * base, const pdb_timer_config_t * configPtr )`

This function configure the PDB's basic timer.

Parameters

|                  |                                                                                 |
|------------------|---------------------------------------------------------------------------------|
| <i>base</i>      | Register base address for the module.                                           |
| <i>configPtr</i> | Pointer to configuration structure, see to " <code>pdb_timer_config_t</code> ". |

Returns

Execution status.

### 28.2.4.3 `static void PDB_HAL_SetSoftTriggerCmd ( PDB_Type * base ) [inline], [static]`

If enabled, this function triggers the DAC by using software.

Parameters

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

### 28.2.4.4 `static void PDB_HAL_Enable ( PDB_Type * base ) [inline], [static]`

This function switches on to enable the PDB module.

Parameters

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

### 28.2.4.5 `static void PDB_HAL_Disable ( PDB_Type * base ) [inline], [static]`

This function switches to disable the PDB module.



Parameters

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

**28.2.4.6 static bool PDB\_HAL\_GetTimerIntFlag ( PDB\_Type \* *base* ) [inline], [static]**

This function gets the PDB delay interrupt flag.

Parameters

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

Returns

Flat status, true if the flag is set.

**28.2.4.7 static void PDB\_HAL\_ClearTimerIntFlag ( PDB\_Type \* *base* ) [inline], [static]**

This function clears PDB delay interrupt flag.

Parameters

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

Returns

Flat status, true if the flag is set.

**28.2.4.8 static void PDB\_HAL\_SetLoadValuesCmd ( PDB\_Type \* *base* ) [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.

## PDB HAL driver

Parameters

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

**28.2.4.9 static void PDB\_HAL\_SetTimerModulusValue ( PDB\_Type \* *base*, 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>base</i>  | Register base address for the module.             |
| <i>value</i> | The setting value of upper limit for PDB counter. |

**28.2.4.10 static uint32\_t PDB\_HAL\_GetTimerValue ( PDB\_Type \* *base* ) [inline], [static]**

This function gets the PDB counter value of PDB timer.

Parameters

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

Returns

The current counter value.

**28.2.4.11 static void PDB\_HAL\_SetValueForTimerInterrupt ( PDB\_Type \* *base*, 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>base</i> | Register base address for the module. |
|-------------|---------------------------------------|

|              |                                                                 |
|--------------|-----------------------------------------------------------------|
| <i>value</i> | The setting value for interrupt delay milestone of PDB counter. |
|--------------|-----------------------------------------------------------------|

**28.2.4.12 void PDB\_HAL\_SetAdcPreTriggerBackToBackEnable ( PDB\_Type \* *base*, uint32\_t *chn*, uint32\_t *preChnMask*, bool *enable* )**

This function switches to enable the pre-trigger back-to-back mode.

Parameters

|                   |                                           |
|-------------------|-------------------------------------------|
| <i>base</i>       | Register base address for the module.     |
| <i>chn</i>        | ADC instance index for trigger.           |
| <i>preChnMask</i> | ADC channel group index mask for trigger. |
| <i>enable</i>     | Switcher to assert the feature.           |

**28.2.4.13 void PDB\_HAL\_SetAdcPreTriggerOutputEnable ( PDB\_Type \* *base*, uint32\_t *chn*, uint32\_t *preChnMask*, bool *enable* )**

This function switches to enable pre-trigger output.

Parameters

|                   |                                           |
|-------------------|-------------------------------------------|
| <i>base</i>       | Register base address for the module.     |
| <i>chn</i>        | ADC instance index for trigger.           |
| <i>preChnMask</i> | ADC channel group index mask for trigger. |
| <i>enable</i>     | Switcher to assert the feature.           |

**28.2.4.14 void PDB\_HAL\_SetAdcPreTriggerEnable ( PDB\_Type \* *base*, uint32\_t *chn*, uint32\_t *preChnMask*, bool *enable* )**

This function switches to enable the pre-trigger.

Parameters

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

## PDB HAL driver

|                   |                                           |
|-------------------|-------------------------------------------|
| <i>chn</i>        | ADC instance index for trigger.           |
| <i>preChnMask</i> | ADC channel group index mask for trigger. |
| <i>enable</i>     | Switcher to assert the feature.           |

### 28.2.4.15 **static uint32\_t PDB\_HAL\_GetAdcPreTriggerFlags ( PDB\_Type \* *base*, uint32\_t *chn*, uint32\_t *preChnMask* ) [inline], [static]**

This function gets the flag which indicates the PDB counter has reached the pre-trigger delay value.

Parameters

|                   |                                           |
|-------------------|-------------------------------------------|
| <i>base</i>       | Register base address for the module.     |
| <i>chn</i>        | ADC instance index for trigger.           |
| <i>preChnMask</i> | ADC channel group index mask for trigger. |

Returns

Flag mask. Indicated bit would be 1 if the event is asserted.

### 28.2.4.16 **void PDB\_HAL\_ClearAdcPreTriggerFlags ( PDB\_Type \* *base*, uint32\_t *chn*, uint32\_t *preChnMask* )**

This function clears the flag which indicates that the PDB counter has reached the pre-trigger delay value.

Parameters

|                   |                                           |
|-------------------|-------------------------------------------|
| <i>base</i>       | Register base address for the module.     |
| <i>chn</i>        | ADC instance index for trigger.           |
| <i>preChnMask</i> | ADC channel group index mask for trigger. |

### 28.2.4.17 **static uint32\_t PDB\_HAL\_GetAdcPreTriggerSeqErrFlags ( PDB\_Type \* *base*, uint32\_t *chn*, uint32\_t *preChnMask* ) [inline], [static]**

This function gets the flag which indicates whether a sequence error is detected.

Parameters

|                   |                                           |
|-------------------|-------------------------------------------|
| <i>base</i>       | Register base address for the module.     |
| <i>chn</i>        | ADC instance index for trigger.           |
| <i>preChnMask</i> | ADC channel group index mask for trigger. |

Returns

Flag mask. Indicated bit would be 1 if the event is asserted.

**28.2.4.18 void PDB\_HAL\_ClearAdcPreTriggerSeqErrFlags ( PDB\_Type \* *base*, uint32\_t *chn*, uint32\_t *preChnMask* )**

This function clears the flag which indicates that the sequence error has been detected.

Parameters

|                   |                                           |
|-------------------|-------------------------------------------|
| <i>base</i>       | Register base address for the module.     |
| <i>chn</i>        | ADC instance index for trigger.           |
| <i>preChnMask</i> | ADC channel group index mask for trigger. |

**28.2.4.19 void PDB\_HAL\_SetAdcPreTriggerDelayValue ( PDB\_Type \* *base*, uint32\_t *chn*, uint32\_t *preChn*, uint32\_t *value* )**

This function sets the pre-trigger delay value.

Parameters

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

**28.2.4.20 static void PDB\_HAL\_SetDacExtTriggerInputEnable ( PDB\_Type \* *base*, uint32\_t *dacChn*, bool *enable* ) [inline], [static]**

This function switches to enable the DAC external trigger input.

## PDB HAL driver

### Parameters

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

#### 28.2.4.21 `static void PDB_HAL_SetDacIntervalTriggerEnable ( PDB_Type * base, uint32_t dacChn, bool enable ) [inline], [static]`

This function switches to enable the DAC external trigger input.

### Parameters

|               |                                       |
|---------------|---------------------------------------|
| <i>base</i>   | Register base address for the module. |
| <i>dacChn</i> | DAC instance index for trigger.       |
| <i>enable</i> | Switcher to assert the feature.       |

#### 28.2.4.22 `static void PDB_HAL_SetDacIntervalValue ( PDB_Type * base, uint32_t dacChn, uint32_t value ) [inline], [static]`

This function sets the interval value for the DAC trigger.

### Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | Register base address for the module.   |
| <i>dacChn</i> | DAC instance index for trigger.         |
| <i>value</i>  | Setting value for DAC trigger interval. |

#### 28.2.4.23 `void PDB_HAL_SetCmpPulseOutEnable ( PDB_Type * base, uint32_t pulseChnMask, bool enable )`

This function switches to enable the pulse-out trigger.

### Parameters

---

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>base</i>         | Register base address for the module.     |
| <i>pulseChnMask</i> | Pulse-out channle index mask for trigger. |
| <i>enable</i>       | Switcher to assert the feature.           |

**28.2.4.24 static void PDB\_HAL\_SetCmpPulseOutDelayForHigh ( PDB\_Type \* *base*, uint32\_t *pulseChn*, uint32\_t *value* ) [inline], [static]**

This function sets the counter delay value for the pulse-out goes high.

Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>base</i>     | Register base address for the module. |
| <i>pulseChn</i> | Pulse-out channel index for trigger.  |
| <i>value</i>    | Setting value for PDB delay .         |

**28.2.4.25 static void PDB\_HAL\_SetCmpPulseOutDelayForLow ( PDB\_Type \* *base*, uint32\_t *pulseChn*, uint32\_t *value* ) [inline], [static]**

This function sets the counter delay value for the pulse-out goes low.

Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>base</i>     | Register base address for the module. |
| <i>pulseChn</i> | Pulse-out channel index for trigger.  |
| <i>value</i>    | Setting value for PDB delay .         |

## PDB Peripheral driver

### 28.3 PDB Peripheral driver

#### 28.3.1 Overview

This section describes the programming interface of the PDB Peripheral driver. 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.

#### 28.3.2 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.

#### 28.3.3 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 of "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.

The basic timing/counting step is set when initializing the main PDB counter:

The basic timing/counting step =  $F\_BusClkHz / \text{pdb\_timer\_config\_t.clkPreDiv} / \text{pdb\_timer\_config\_t.clkPreMultFactor}$

The  $F\_BusClkHz$  is the frequency of bus clock in Hertz. The "clkPreDiv" and "clkPreMultFactor" are in the [pdb\\_timer\\_config\\_t](#) structure. All triggering milestones are based on this step.

#### 28.3.4 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 limit, 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 limit for the DAC counter. Once the counter reaches the upper limit, it turns the DAC counter to zero and counts again. When the DAC counter hits the upper limit, 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 limit, 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 "pdb_test.h"

static volatile uint32_t gPdbIntCounter = 0U;
static volatile uint32_t gPdbInstance = 0U;

static void PDB_ISR_Counter(void);

void PDB_TEST_NormalTimer(uint32_t instance)
{
 pdb_timer_config_t PdbTimerConfig;

 PdbTimerConfig.loadValueMode = kPdbLoadValueImmediately;
 PdbTimerConfig.seqErrIntEnable = false;
 PdbTimerConfig.clkPreDiv = kPdbClkPreDivBy8;
 PdbTimerConfig.clkPreMultFactor = kPdbClkPreMultFactorAs40;
 PdbTimerConfig.triggerInput = kPdbSoftTrigger;
 PdbTimerConfig.continuousModeEnable = true;
 PdbTimerConfig.dmaEnable = false;
 PdbTimerConfig.intEnable = true;
 PDB_DRV_Init(instance, &PdbTimerConfig);

 PDB_DRV_SetTimerModulusValue(instance, 0xFFFFU);
 PDB_DRV_SetValueForTimerInterrupt(instance, 0xFFU);
 PDB_DRV_LoadValuesCmd(instance);
 gPdbIntCounter = 0U;

 gPdbInstance = instance;
 PDB_TEST_InstallCallback(instance, PDB_ISR_Counter);

 PDB_DRV_SoftTriggerCmd(instance);
 while (gPdbIntCounter < 20U) {}
 PRINTF("PDB Timer's delay interrupt generated.\r\n");
}
```

## PDB Peripheral driver

```

PDB_DRV_Deinit(instance);

PRINTF("OK.\r\n");
}

static void PDB_ISR_Counter(void)
{
 if (gPdbIntCounter >= 0xFFFFU)
 {
 gPdbIntCounter = 0U;
 }
 else
 {
 gPdbIntCounter++;
 }
}

```

### Trigger for ADC module:

```

#include "pdb_test.h"

// pdb_test_adc_trigger.c //

void PDB_TEST_AdcPreTrigger(uint32_t instance)
{
 pdb_timer_config_t PdbTimerConfig;
 pdb_adc_pretrigger_config_t PdbAdcPreTriggerConfig;

 PdbTimerConfig.loadValueMode = kPdbLoadValueImmediately;
 PdbTimerConfig.seqErrIntEnable = false;
 PdbTimerConfig.clkPreDiv = kPdbClkPreDivBy8;
 PdbTimerConfig.clkPreMultFactor = kPdbClkPreMultFactorAs40;
 PdbTimerConfig.triggerInput = kPdbSoftTrigger;
 PdbTimerConfig.continuousModeEnable = false;
 PdbTimerConfig.dmaEnable = false;
 PdbTimerConfig.intEnable = false;
 PDB_DRV_Init(instance, &PdbTimerConfig);

 PdbAdcPreTriggerConfig.adcPreTriggerIdx = 0U;
 PdbAdcPreTriggerConfig.preTriggerEnable = true;
 PdbAdcPreTriggerConfig.preTriggerOutputEnable = true;
 PdbAdcPreTriggerConfig.preTriggerBackToBackEnable = false;
 PDB_DRV_ConfigAdcPreTrigger(instance, 0U, &PdbAdcPreTriggerConfig);

 PDB_DRV_SetTimerModulusValue(instance, 0xFFFFU);
 PDB_DRV_SetAdcPreTriggerDelayValue(instance, 0U, 0U, 0xFFFFU);
 PDB_DRV_LoadValuesCmd(instance);

 PDB_DRV_SoftTriggerCmd(instance);
 while (1U != PDB_DRV_GetAdcPreTriggerFlags(instance, 0U, 1U)) {}
 PDB_DRV_ClearAdcPreTriggerFlags(instance, 0U, 1U);
 PRINTF("PDB ADC PreTrigger generated.\r\n");

 PDB_DRV_Deinit(instance);

 PRINTF("OK.\r\n");
}

```

## Data Structures

- struct `pdb_adc_pretrigger_config_t`  
*Defines the type of structure for configuring ADC's pre\_trigger. [More...](#)*

- struct [pdb\\_dac\\_interval\\_config\\_t](#)  
*Defines the type of flag for PDB pre-trigger events. [More...](#)*

## Functions

- [pdb\\_status\\_t PDB\\_DRV\\_Init](#) (uint32\_t instance, const [pdb\\_timer\\_config\\_t](#) \*userConfigPtr)  
*Initializes the PDB counter and triggers input.*
- [pdb\\_status\\_t 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\\_GetTimerValue](#) (uint32\_t instance)  
*Gets the current counter value in the PDB module.*
- bool [PDB\\_DRV\\_GetTimerIntFlag](#) (uint32\_t instance)  
*Gets the PDB interrupt flag.*
- void [PDB\\_DRV\\_ClearTimerIntFlag](#) (uint32\_t instance)  
*Clears the interrupt flag.*
- void [PDB\\_DRV\\_LoadValuesCmd](#) (uint32\_t instance)  
*Executes the command of loading values.*
- void [PDB\\_DRV\\_SetTimerModulusValue](#) (uint32\_t instance, uint32\_t value)  
*Sets the value of timer modulus.*
- void [PDB\\_DRV\\_SetValueForTimerInterrupt](#) (uint32\_t instance, uint32\_t value)  
*Sets the value for the timer interrupt.*
- [pdb\\_status\\_t PDB\\_DRV\\_ConfigAdcPreTrigger](#) (uint32\_t instance, uint32\_t chn, const [pdb\\_adc\\_pretrigger\\_config\\_t](#) \*configPtr)  
*Configures the ADC pre\_trigger in the PDB module.*
- uint32\_t [PDB\\_DRV\\_GetAdcPreTriggerFlags](#) (uint32\_t instance, uint32\_t chn, uint32\_t preChnMask)  
*Gets the ADC pre\_trigger flag in the PDB module.*
- void [PDB\\_DRV\\_ClearAdcPreTriggerFlags](#) (uint32\_t instance, uint32\_t chn, uint32\_t preChnMask)  
*Clears the ADC pre\_trigger flag in the PDB module.*
- uint32\_t [PDB\\_DRV\\_GetAdcPreTriggerSeqErrFlags](#) (uint32\_t instance, uint32\_t chn, uint32\_t preChnMask)  
*Gets the ADC pre\_trigger flag in the PDB module.*
- void [PDB\\_DRV\\_ClearAdcPreTriggerSeqErrFlags](#) (uint32\_t instance, uint32\_t chn, uint32\_t preChnMask)  
*Clears the ADC pre\_trigger flag in the PDB module.*
- void [PDB\\_DRV\\_SetAdcPreTriggerDelayValue](#) (uint32\_t instance, uint32\_t chn, uint32\_t preChn, uint32\_t value)  
*Sets the ADC pre\_trigger delay value in the PDB module.*
- [pdb\\_status\\_t PDB\\_DRV\\_ConfigDacInterval](#) (uint32\_t instance, uint32\_t dacChn, const [pdb\\_dac\\_interval\\_config\\_t](#) \*configPtr)  
*Configures the DAC interval in the PDB module.*
- void [PDB\\_DRV\\_SetDacIntervalValue](#) (uint32\_t instance, uint32\_t dacChn, uint32\_t value)  
*Sets the DAC interval value in the PDB module.*
- void [PDB\\_DRV\\_SetCmpPulseOutEnable](#) (uint32\_t instance, uint32\_t pulseChnMask, bool enable)  
*Switches on/off the CMP pulse out in the PDB module.*
- void [PDB\\_DRV\\_SetCmpPulseOutDelayForHigh](#) (uint32\_t instance, uint32\_t pulseChn, uint32\_t value)

## PDB Peripheral driver

- Sets the CMP pulse out delay value for high in the PDB module.*
- void [PDB\\_DRV\\_SetCmpPulseOutDelayForLow](#) (uint32\_t instance, uint32\_t pulseChn, uint32\_t value)

*Sets the CMP pulse out delay value for low in the PDB module.*

## Variables

- PDB\_Type \*const [g\\_pdbBase](#) []

*Table of base addresses for PDB instances.*
- const IRQn\_Type [g\\_pdbIrqId](#) [PDB\_INSTANCE\_COUNT]

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

## 28.3.5 Data Structure Documentation

### 28.3.5.1 struct `pdb_adc_pretrigger_config_t`

#### Data Fields

- uint32\_t [adcPreTriggerIdx](#)

*Setting pre\_trigger's index.*
- bool [preTriggerEnable](#)

*Enable the pre\_trigger.*
- bool [preTriggerOutputEnable](#)

*Enable the pre\_trigger output.*
- bool [preTriggerBackToBackEnable](#)

*Enable the back to back mode for ADC pre\_trigger.*

#### 28.3.5.1.0.51 Field Documentation

28.3.5.1.0.51.1 uint32\_t `pdb_adc_pretrigger_config_t::adcPreTriggerIdx`

28.3.5.1.0.51.2 bool `pdb_adc_pretrigger_config_t::preTriggerEnable`

### 28.3.5.2 struct `pdb_dac_interval_config_t`

#### Data Fields

- bool [intervalTriggerEnable](#)

*Enable the DAC interval trigger.*
- bool [extTriggerInputEnable](#)

*Enable DAC external trigger input .*

**28.3.5.2.0.52 Field Documentation**

**28.3.5.2.0.52.1** bool `pdb_dac_interval_config_t::intervalTriggerEnable`

**28.3.6 Function Documentation**

**28.3.6.1** `pdb_status_t PDB_DRV_Init ( uint32_t instance, const pdb_timer_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 " <code>pdb_user_config_t</code> ". |

Returns

Execution status.

**28.3.6.2** `pdb_status_t 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. |
|-----------------|------------------|

Returns

Execution status.

**28.3.6.3** `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. |
|-----------------|------------------|

### 28.3.6.4 uint32\_t PDB\_DRV\_GetTimerValue ( uint32\_t *instance* )

This function gets the current counter value.

Parameters

|                 |                  |
|-----------------|------------------|
| <i>instance</i> | PDB instance ID. |
|-----------------|------------------|

Returns

Current PDB counter value.

### 28.3.6.5 bool PDB\_DRV\_GetTimerIntFlag ( 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.

### 28.3.6.6 void PDB\_DRV\_ClearTimerIntFlag ( uint32\_t *instance* )

This function clears the interrupt flag.

Parameters

|                 |                  |
|-----------------|------------------|
| <i>instance</i> | PDB instance ID. |
|-----------------|------------------|

### 28.3.6.7 void PDB\_DRV\_LoadValuesCmd ( uint32\_t *instance* )

This function executes the command of loading values.

Parameters

|                 |                  |
|-----------------|------------------|
| <i>instance</i> | PDB instance ID. |
| <i>value</i>    | Setting value.   |

### 28.3.6.8 void PDB\_DRV\_SetTimerModulusValue ( uint32\_t *instance*, uint32\_t *value* )

This function sets the value of timer modulus.

Parameters

|                 |                  |
|-----------------|------------------|
| <i>instance</i> | PDB instance ID. |
| <i>value</i>    | Setting value.   |

### 28.3.6.9 void PDB\_DRV\_SetValueForTimerInterrupt ( uint32\_t *instance*, uint32\_t *value* )

This function sets the value for the timer interrupt.

Parameters

|                 |                  |
|-----------------|------------------|
| <i>instance</i> | PDB instance ID. |
| <i>value</i>    | Setting value.   |

### 28.3.6.10 pdb\_status\_t PDB\_DRV\_ConfigAdcPreTrigger ( uint32\_t *instance*, uint32\_t *chn*, const pdb\_adc\_pretrigger\_config\_t \* *configPtr* )

This function configures the ADC pre\_trigger in the PDB module.

Parameters

|                  |                                                                                     |
|------------------|-------------------------------------------------------------------------------------|
| <i>instance</i>  | PDB instance ID.                                                                    |
| <i>chn</i>       | ADC channel.                                                                        |
| <i>configPtr</i> | Pointer to the user configuration structure. See the "pdb_adc_pretrigger_config_t". |

Returns

Execution status.

## PDB Peripheral driver

**28.3.6.11** `uint32_t PDB_DRV_GetAdcPreTriggerFlags ( uint32_t instance, uint32_t chn,  
uint32_t preChnMask )`

This function gets the ADC pre\_trigger flags in the PDB module.



Parameters

|                   |                                |
|-------------------|--------------------------------|
| <i>instance</i>   | PDB instance ID.               |
| <i>chn</i>        | ADC channel.                   |
| <i>preChnMask</i> | ADC pre_trigger channels mask. |

Returns

Assertion of indicated flag.

**28.3.6.12 void PDB\_DRV\_ClearAdcPreTriggerFlags ( uint32\_t *instance*, uint32\_t *chn*, uint32\_t *preChnMask* )**

This function clears the ADC pre\_trigger flags in the PDB module.

Parameters

|                   |                                |
|-------------------|--------------------------------|
| <i>instance</i>   | PDB instance ID.               |
| <i>chn</i>        | ADC channel.                   |
| <i>preChnMask</i> | ADC pre_trigger channels mask. |

**28.3.6.13 uint32\_t PDB\_DRV\_GetAdcPreTriggerSeqErrFlags ( uint32\_t *instance*, uint32\_t *chn*, uint32\_t *preChnMask* )**

This function gets the ADC pre\_trigger flags in the PDB module.

Parameters

|                   |                                |
|-------------------|--------------------------------|
| <i>instance</i>   | PDB instance ID.               |
| <i>chn</i>        | ADC channel.                   |
| <i>preChnMask</i> | ADC pre_trigger channels mask. |

Returns

Assertion of indicated flag.

**28.3.6.14 void PDB\_DRV\_ClearAdcPreTriggerSeqErrFlags ( uint32\_t *instance*, uint32\_t *chn*, uint32\_t *preChnMask* )**

This function clears the ADC pre\_trigger sequence error flags in the PDB module.

## PDB Peripheral driver

Parameters

|                   |                                |
|-------------------|--------------------------------|
| <i>instance</i>   | PDB instance ID.               |
| <i>chn</i>        | ADC channel.                   |
| <i>preChnMask</i> | ADC pre_trigger channels mask. |

### 28.3.6.15 void PDB\_DRV\_SetAdcPreTriggerDelayValue ( uint32\_t *instance*, uint32\_t *chn*, uint32\_t *preChn*, uint32\_t *value* )

This function sets Set the ADC pre\_trigger delay value in the PDB module.

Parameters

|                 |                  |
|-----------------|------------------|
| <i>instance</i> | PDB instance ID. |
| <i>chn</i>      | ADC channel.     |
| <i>preChn</i>   | ADC pre_channel. |
| <i>value</i>    | Setting value.   |

### 28.3.6.16 pdb\_status\_t PDB\_DRV\_ConfigDacInterval ( uint32\_t *instance*, uint32\_t *dacChn*, const pdb\_dac\_interval\_config\_t \* *configPtr* )

This function configures the DAC interval in the PDB module.

Parameters

|                  |                                                                                   |
|------------------|-----------------------------------------------------------------------------------|
| <i>instance</i>  | PDB instance ID.                                                                  |
| <i>dacChn</i>    | DAC channel.                                                                      |
| <i>configPtr</i> | Pointer to the user configuration structure. See the "pdb_dac_interval_config_t". |

Returns

Execution status.

### 28.3.6.17 void PDB\_DRV\_SetDacIntervalValue ( uint32\_t *instance*, uint32\_t *dacChn*, uint32\_t *value* )

This function sets the DAC interval value in the PDB module.

Parameters

|                 |                  |
|-----------------|------------------|
| <i>instance</i> | PDB instance ID. |
| <i>dacChn</i>   | DAC channel.     |
| <i>value</i>    | Setting value.   |

### 28.3.6.18 void PDB\_DRV\_SetCmpPulseOutEnable ( uint32\_t *instance*, uint32\_t *pulseChnMask*, bool *enable* )

This function switches the CMP pulse on/off in the PDB module.

Parameters

|                     |                                 |
|---------------------|---------------------------------|
| <i>instance</i>     | PDB instance ID.                |
| <i>pulseChnMask</i> | Pulse channel mask.             |
| <i>enable</i>       | Switcher to assert the feature. |

### 28.3.6.19 void PDB\_DRV\_SetCmpPulseOutDelayForHigh ( uint32\_t *instance*, uint32\_t *pulseChn*, uint32\_t *value* )

This function sets the CMP pulse out delay value for high in the PDB module.

Parameters

|                 |                  |
|-----------------|------------------|
| <i>instance</i> | PDB instance ID. |
| <i>pulseChn</i> | Pulse channel.   |
| <i>value</i>    | Setting value.   |

### 28.3.6.20 void PDB\_DRV\_SetCmpPulseOutDelayForLow ( uint32\_t *instance*, uint32\_t *pulseChn*, uint32\_t *value* )

This function sets the CMP pulse out delay value for low in the PDB module.

Parameters

---

## PDB Peripheral driver

|                 |                  |
|-----------------|------------------|
| <i>instance</i> | PDB instance ID. |
| <i>pulseChn</i> | Pulse channel.   |
| <i>value</i>    | Setting value.   |

### 28.3.7 Variable Documentation

28.3.7.1 `PDB_Type* const g_pdbBase[]`

28.3.7.2 `const IRQn_Type g_pdbIrqId[PDB_INSTANCE_COUNT]`

## Chapter 29

# Periodic Interrupt Timer (PIT)

### 29.1 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

### 29.2 PIT HAL driver

#### 29.2.1 Overview

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

#### Enumerations

- enum [pit\\_status\\_t](#)  
*Error codes for PIT driver.*

#### Initialization

- static void [PIT\\_HAL\\_Enable](#) (PIT\_Type \*base)  
*Enables the PIT module.*
- static void [PIT\\_HAL\\_Disable](#) (PIT\_Type \*base)  
*Disables the PIT module.*
- static void [PIT\\_HAL\\_SetTimerRunInDebugCmd](#) (PIT\_Type \*base, bool timerRun)  
*Configures the timers to continue running or to stop in debug mode.*

#### Timer Start and Stop

- static void [PIT\\_HAL\\_StartTimer](#) (PIT\_Type \*base, uint32\_t channel)  
*Starts the timer counting.*
- static void [PIT\\_HAL\\_StopTimer](#) (PIT\_Type \*base, uint32\_t channel)  
*Stops the timer from counting.*
- static bool [PIT\\_HAL\\_IsTimerRunning](#) (PIT\_Type \*base, uint32\_t channel)  
*Checks to see whether the current timer is started or not.*

#### Timer Period

- static void [PIT\\_HAL\\_SetTimerPeriodByCount](#) (PIT\_Type \*base, uint32\_t channel, uint32\_t count)  
*Sets the timer period in units of count.*
- static uint32\_t [PIT\\_HAL\\_GetTimerPeriodByCount](#) (PIT\_Type \*base, uint32\_t channel)  
*Returns the current timer period in units of count.*
- static uint32\_t [PIT\\_HAL\\_ReadTimerCount](#) (PIT\_Type \*base, uint32\_t channel)  
*Reads the current timer counting value.*

#### Interrupt

- static void [PIT\\_HAL\\_SetIntCmd](#) (PIT\_Type \*base, uint32\_t channel, bool enable)  
*Enables or disables the timer interrupt.*
- static bool [PIT\\_HAL\\_GetIntCmd](#) (PIT\_Type \*base, uint32\_t channel)  
*Checks whether the timer interrupt is enabled or not.*

- static void **PIT\_HAL\_ClearIntFlag** (PIT\_Type \*base, uint32\_t channel)  
*Clears the timer interrupt flag.*
- static bool **PIT\_HAL\_IsIntPending** (PIT\_Type \*base, uint32\_t channel)  
*Reads the current timer timeout flag.*

## 29.2.2 Enumeration Type Documentation

### 29.2.2.1 enum pit\_status\_t

## 29.2.3 Function Documentation

### 29.2.3.1 static void PIT\_HAL\_Enable ( PIT\_Type \* *base* ) [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.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current PIT instance. |
|-------------|----------------------------------------|

### 29.2.3.2 static void PIT\_HAL\_Disable ( PIT\_Type \* *base* ) [inline], [static]

This function disables all PIT timer clocks (Note: it does not affect the SIM clock gating control).

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current PIT instance. |
|-------------|----------------------------------------|

### 29.2.3.3 static void PIT\_HAL\_SetTimerRunInDebugCmd ( PIT\_Type \* *base*, 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

---

## PIT HAL driver

|                 |                                                                                                                                                                                |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>     | Base address for current PIT instance.                                                                                                                                         |
| <i>timerRun</i> | Timers run or stop in debug mode. <ul style="list-style-type: none"> <li>• true: Timers continue to run in debug mode.</li> <li>• false: Timers stop in debug mode.</li> </ul> |

### 29.2.3.4 static void PIT\_HAL\_StartTimer ( PIT\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

After calling this function, timers load the start value as specified by the function [PIT\\_HAL\\_SetTimerPeriodByCount\(PIT\\_Type \\* base, uint32\\_t channel, uint32\\_t count\)](#), 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>base</i>    | Base address for current PIT instance. |
| <i>channel</i> | Timer channel number                   |

### 29.2.3.5 static void PIT\_HAL\_StopTimer ( PIT\_Type \* *base*, 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>base</i>    | Base address for current PIT instance. |
| <i>channel</i> | Timer channel number                   |

### 29.2.3.6 static bool PIT\_HAL\_IsTimerRunning ( PIT\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

Parameters



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

### 29.2.3.7 **static void PIT\_HAL\_SetTimerPeriodByCount ( PIT\_Type \* *base*, 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

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current PIT instance. |
| <i>channel</i> | Timer channel number                   |
| <i>count</i>   | Timer period in units of count         |

### 29.2.3.8 **static uint32\_t PIT\_HAL\_GetTimerPeriodByCount ( PIT\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current PIT instance. |
| <i>channel</i> | Timer channel number                   |

Returns

Timer period in units of count

### 29.2.3.9 **static uint32\_t PIT\_HAL\_ReadTimerCount ( PIT\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

This function returns the real-time timer counting value, in a range from 0 to a timer period.

## PIT HAL driver

### Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current PIT instance. |
| <i>channel</i> | Timer channel number                   |

### Returns

Current timer counting value

**29.2.3.10 static void PIT\_HAL\_SetIntCmd ( PIT\_Type \* *base*, 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>base</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"> <li>• true: Generate interrupt when timer counts to 0.</li> <li>• false: No interrupt is generated.</li> </ul> |

**29.2.3.11 static bool PIT\_HAL\_GetIntCmd ( PIT\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

### Parameters

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

**29.2.3.12 static void PIT\_HAL\_ClearIntFlag ( PIT\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

This function clears the timer interrupt flag after a timeout event occurs.

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current PIT instance. |
| <i>channel</i> | Timer channel number                   |

**29.2.3.13 static bool PIT\_HAL\_IsIntPending ( PIT\_Type \* *base*, uint32\_t *channel* )  
[inline], [static]**

Every time the timer counts to 0, this flag is set.

Parameters

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

## PIT Peripheral driver

### 29.3 PIT Peripheral driver

#### 29.3.1 Overview

This section describes the programming interface of the PIT Peripheral driver. The PIT driver configures PIT timers and initializes and configures timer periods.

#### 29.3.2 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);
```

#### 29.3.3 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.

### 29.3.4 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.

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...](#)*

### Variables

- PIT\_Type \*const `g_pitBase []`  
*Table of base addresses for pit instances.*

### Initialization and Shutdown

- `pit_status_t PIT_DRV_Init` (uint32\_t instance, bool isRunInDebug)  
*Initializes the PIT module.*
- `pit_status_t 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)  
*Sets the timer period in microseconds.*
- uint32\_t `PIT_DRV_GetTimerPeriodByUs` (uint32\_t instance, uint32\_t channel)  
*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)

## PIT Peripheral driver

- *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.*

## Interrupt

- void [PIT\\_DRV\\_ClearIntFlag](#) (uint32\_t instance, uint32\_t channel)  
*Clears the timer interrupt flag.*
- bool [PIT\\_DRV\\_IsIntPending](#) (uint32\_t instance, uint32\_t channel)  
*Reads the current timer timeout flag.*

## 29.3.5 Data Structure Documentation

### 29.3.5.1 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 is bypassed inside the [PIT\\_DRV\\_InitChannel\(\)](#) function.

### Data Fields

- bool [isInterruptEnabled](#)  
*Timer interrupt 0-disable/1-enable.*
- uint32\_t [periodUs](#)  
*Timer period in unit of microseconds.*

## 29.3.6 Function Documentation

### 29.3.6.1 pit\_status\_t PIT\_DRV\_Init ( uint32\_t instance, bool isRunInDebug )

Call this function before calling all the other PIT driver functions. This function un-gates the PIT clock and enables the PIT module. The `isRunInDebug` passed into function affects all timer channels.

## Parameters

|                     |                                                                                                                                                                                |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>     | PIT module instance number.                                                                                                                                                    |
| <i>isRunInDebug</i> | Timers run or stop in debug mode. <ul style="list-style-type: none"> <li>• true: Timers continue to run in debug mode.</li> <li>• false: Timers stop in debug mode.</li> </ul> |

## Returns

Error or success status returned by API.

**29.3.6.2 pit\_status\_t 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. |
|-----------------|-----------------------------|

## Returns

Error or success status returned by API.

**29.3.6.3 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,
 // In unit of microseconds.
 .periodUs = 1000,
};
```

## PIT Peripheral driver

Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>instance</i> | PIT module instance number.          |
| <i>channel</i>  | Timer channel number.                |
| <i>config</i>   | PIT channel configuration structure. |

### 29.3.6.4 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.       |

### 29.3.6.5 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.       |

### 29.3.6.6 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. |

**29.3.6.7 uint32\_t PIT\_DRV\_GetTimerPeriodByUs ( uint32\_t instance, uint32\_t channel )**

Parameters

|                 |                             |
|-----------------|-----------------------------|
| <i>instance</i> | PIT module instance number. |
| <i>channel</i>  | Timer channel number.       |

Returns

Timer period in microseconds.

**29.3.6.8 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.

**29.3.6.9 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.

## PIT Peripheral driver

Parameters

|                 |                                |
|-----------------|--------------------------------|
| <i>instance</i> | PIT module instance number.    |
| <i>channel</i>  | Timer channel number           |
| <i>count</i>    | Timer period in units of count |

### 29.3.6.10 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

### 29.3.6.11 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

### 29.3.6.12 void PIT\_DRV\_ClearIntFlag ( uint32\_t *instance*, uint32\_t *channel* )

This function clears the timer interrupt flag after a timeout event occurs.

Parameters

|                 |                             |
|-----------------|-----------------------------|
| <i>instance</i> | PIT module instance number. |
| <i>channel</i>  | Timer channel number        |

**29.3.6.13 bool PIT\_DRV\_IsIntPending ( uint32\_t *instance*, uint32\_t *channel* )**

Every time the timer counts to 0, this flag is set.

Parameters

|                 |                             |
|-----------------|-----------------------------|
| <i>instance</i> | PIT module instance number. |
| <i>channel</i>  | Timer channel number        |

Returns

- Current status of the timeout flag
- true: Timeout has occurred.
  - false: Timeout has not yet occurred.

**29.3.7 Variable Documentation**

**29.3.7.1 PIT\_Type\* const g\_pitBase[]**



---

## Chapter 30

# Pulse Width Modulator A (PWMA/eFlexPWM)

### 30.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the eFlexPWM (PWMA/eFlexPWM) block of Kinetis devices.

#### Modules

- [eFlexPWM Peripheral Driver](#)

## eFlexPWM Peripheral Driver

### 30.2 eFlexPWM Peripheral Driver

#### 30.2.1 Overview

The section describes the programming interface of the eFlexPWM Peripheral driver. The Pulse Width Modulator A (PWMA/eFlexPWM) module contains PWM sub modules, each of which is set up to control a single half-bridge power stage. Fault channel support is provided. This PWM module can generate various switching patterns, including highly sophisticated waveforms. It can be used to control all known Switched Mode Power Supplies (SMPS) topologies.

#### 30.2.2 Initialization

To initialize the PWM driver, call the [PWM\\_DRV\\_Init\(\)](#) function and pass the instance and sub module numbers of the relevant PWM. For instance, to use PWM0, pass a value of 0 to the initialization function. In addition, you should also pass a user configuration structure `pwm_user_config_t`, as shown here:

```
// PWM configuration structure for user
typedef struct PwmUserConfig {
 uint8_t Clock;
 uint8_t OutputTrigger;
 uint8_t Polarity;
 bool isFaultEnable;
} pwm_user_config_t;
```

#### 30.2.3 Generate a PWM signal

The PWM calls the `PWM_DRV_PwmStart()` function to generate a PWM signal. Use this structure to configure different parameters related to this PWM signal:

```
typedef struct FtmPwmParam
typedef struct PwmParam
{
 uint32_t uFrequencyHZ;
 uint32_t uDutyCyclePercent;
}pwm_param_t;
```

HAS TO BE UPDATED The mode options are `kFtmEdgeAlignedPWM`, `kFtmCenterAlignedPWM` and `kFtmCombinedPWM`. For edge mode, the options available are `kFtmHighTrue` and `kFtmLowTrue`. Specify the PWM signal frequency in Hertz and the duty cycle percentage (value between 0-100). When the PWM mode is `kFtmCombinedPWM`, specify a value for `uFirstEdgeDelayPercent` which delays the start of the PWM pulse.

#### Data Structures

- struct [pwm\\_module\\_signal\\_setup\\_t](#)  
Configuration structure for the user to define the PWM signal characteristics. [More...](#)

## Macros

- #define **FLEXPWM\_NO\_PWM\_OUT\_SIGNAL** (0)  
*Used to indicate a particular pin in the submodule does NOT output a PWM signal.*

## Enumerations

- enum **pwm\_signal\_type\_t** {  
    **kFlexPwmSignedCenterAligned** = 0U,  
    **kFlexPwmCenterAligned**,  
    **kFlexPwmSignedEdgeAligned**,  
    **kFlexPwmEdgeAligned** }  
*PWM Signal Type options.*

## Functions

- **pwm\_status\_t PWM\_DRV\_Init** (uint32\_t instance)  
*Initializes the PWM module.*
- void **PWM\_DRV\_Deinit** (uint32\_t instance)  
*Shuts down the PWM driver.*
- **pwm\_status\_t PWM\_DRV\_SetupPwm** (uint32\_t instance, **pwm\_module\_t** subModule, **pwm\_module\_setup\_t** \*moduleSetupParams, **pwm\_module\_signal\_setup\_t** \*signalParams)  
*Sets up the PWM signals from the FlewPWM module.*
- void **PWM\_DRV\_UpdatePwmSignal** (uint32\_t instance, **pwm\_module\_t** subModule, **pwm\_module\_signal\_setup\_t** \*signalParams)  
*Updates the PWM signal settings.*
- void **PWM\_DRV\_SetTriggerCmd** (uint32\_t instance, **pwm\_module\_t** subModule, **pwm\_val\_regs\_t** trigger, bool activate)  
*Enables or disables the PWM output trigger.*
- void **PWM\_DRV\_SetTriggerVal** (uint32\_t instance, **pwm\_module\_t** subModule, **pwm\_val\_regs\_t** trigger, uint16\_t triggerVal)  
*Sets the PWM trigger value.*
- void **PWM\_DRV\_SetupFault** (uint32\_t instance, **pwm\_module\_t** subModule, **pwm\_fault\_input\_t** faultNum, **pwm\_fault\_setup\_t** \*faultParams, bool pwmA, bool pwmB, bool pwmX)  
*Sets up the PWM fault.*
- void **PWM\_DRV\_CounterStart** (uint32\_t instance, uint8\_t value)  
*Starts the PWM counter.*
- void **PWM\_DRV\_CounterStop** (uint32\_t instance, uint8\_t value)  
*Stops the PWM counter.*
- void **PWM\_DRV\_SetExternalClkFreq** (uint32\_t instance, uint32\_t externalClkFreq)  
*Provides the frequency of the external clock source.*

## Variables

- **PWM\_Type** \*const **g\_pwmBase** [PWM\_INSTANCE\_COUNT]  
*Table of base addresses for PWM instances.*

## eFlexPWM Peripheral Driver

- const IRQn\_Type [g\\_pwmCmpIrqId](#) [FSL\_FEATURE\_PWM\_CMP\_INT\_HANDLER\_COUNT]  
Table to save PWM IRQ enumeration numbers defined in CMSIS header file.

### 30.2.4 Data Structure Documentation

#### 30.2.4.1 struct pwm\_module\_signal\_setup\_t

##### Data Fields

- uint32\_t [pwmPeriod](#)  
*PWM period specified in microseconds.*
- [pwm\\_signal\\_type\\_t](#) [pwmType](#)  
*PWM type, edge or center; signed or unsigned.*
- uint32\_t [pwmAPulseWidth](#)  
*PWM A pulse width specified in microseconds.*
- uint32\_t [pwmBPulseWidth](#)  
*PWM B pulse width specified in microseconds.*
- bool [pwmAPolarity](#)  
*true: if output is to be inverted; false: if no output inversion.*
- bool [pwmBPolarity](#)  
*true: if output is to be inverted; false: if no output inversion.*

##### 30.2.4.1.0.53 Field Documentation

###### 30.2.4.1.0.53.1 uint32\_t pwm\_module\_signal\_setup\_t::pwmAPulseWidth

Specify FLEXPWM\_NO\_PWM\_OUT\_SIGNAL if no PWM output on this pin.

###### 30.2.4.1.0.53.2 uint32\_t pwm\_module\_signal\_setup\_t::pwmBPulseWidth

Specify FLEXPWM\_NO\_PWM\_OUT\_SIGNAL if no PWM output on this pin.

### 30.2.5 Enumeration Type Documentation

#### 30.2.5.1 enum pwm\_signal\_type\_t

Enumerator

- kFlexPwmSignedCenterAligned* Signed center-aligned.
- kFlexPwmCenterAligned* Unsigned center-aligned.
- kFlexPwmSignedEdgeAligned* Signed edge-aligned.
- kFlexPwmEdgeAligned* Unsigned edge-aligned.



## 30.2.6 Function Documentation

### 30.2.6.1 `pwm_status_t PWM_DRV_Init ( uint32_t instance )`

Enables the module clocks and interrupts in the interrupt controller.

## eFlexPWM Peripheral Driver

### Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>instance</i> | Instance number of the PWM module. |
|-----------------|------------------------------------|

### Returns

kStatusPwmSuccess means success, otherwise means failed.

### 30.2.6.2 void PWM\_DRV\_Deinit ( uint32\_t *instance* )

This function de-initializes the EflexPWM module and disables the clock for the submodules. Function disables the module-level interrupts.

### Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>instance</i> | Instance number of the PWM module. |
|-----------------|------------------------------------|

### 30.2.6.3 pwm\_status\_t PWM\_DRV\_SetupPwm ( uint32\_t *instance*, pwm\_module\_t *subModule*, pwm\_module\_setup\_t \* *moduleSetupParams*, pwm\_module\_signal\_setup\_t \* *signalParams* )

The function initializes the submodule per the parameters passed in by the user. The function also sets up the value compare registers to match the PWM signal requirements. NOTE: If the deadtime insertion logic is enabled, the pulse period is reduced by the deadtime period specified by the user.

### Parameters

|                          |                                                             |
|--------------------------|-------------------------------------------------------------|
| <i>instance</i>          | Instance number of the PWM module.                          |
| <i>subModule</i>         | The FlexPWM submodule that is configured                    |
| <i>moduleSetupParams</i> | The initialization values used to set up the submodule      |
| <i>signalParams</i>      | Signal parameters which generate the submodules PWM signals |

### Returns

Returns an error if the requested submodule clock is wrong i.e, request for kFlexPwmModule0 for submodule 0 or request for external clock without informing the driver of the external clock frequency by calling PWM\_DVR\_SetExternalClkFreq(). Success otherwise

**30.2.6.4 void PWM\_DRV\_UpdatePwmSignal ( uint32\_t *instance*, pwm\_module\_t *subModule*, pwm\_module\_signal\_setup\_t \* *signalParams* )**

The function updates the PWM signal to the new value that is passed in. NOTE: If the deadtime insertion logic is enabled then the pulse period is reduced by the deadtime period specified by the user.

## eFlexPWM Peripheral Driver

### Parameters

|                     |                                                             |
|---------------------|-------------------------------------------------------------|
| <i>instance</i>     | Instance number of the PWM module.                          |
| <i>subModule</i>    | The FlexPWM submodule that is configured                    |
| <i>signalParams</i> | Signal parameters which generate the submodules PWM signals |

### 30.2.6.5 void PWM\_DRV\_SetTriggerCmd ( uint32\_t *instance*, pwm\_module\_t *subModule*, pwm\_val\_regs\_t *trigger*, bool *activate* )

This function allows the user to enable or disable the PWM trigger. The PWM has 2 triggers. The trigger 0 is activated when the counter matches VAL 0, VAL 2, or VAL 4 register. The trigger 1 is activated when the counter matches VAL 1, VAL 3, or VAL 5.

### Parameters

|                  |                                                |
|------------------|------------------------------------------------|
| <i>instance</i>  | Instance number of the PWM module.             |
| <i>subModule</i> | The FlexPWM submodule that is configured       |
| <i>trigger</i>   | Trigger number that the user wants to activate |
| <i>activate</i>  | Enable or disable the trigger                  |

### 30.2.6.6 void PWM\_DRV\_SetTriggerVal ( uint32\_t *instance*, pwm\_module\_t *subModule*, pwm\_val\_regs\_t *trigger*, uint16\_t *triggerVal* )

This function sets the value in the compare register that generates a trigger. NOTE: User must make sure the value register being modified is not currently used to generate a PWM signal.

### Parameters

|                   |                                          |
|-------------------|------------------------------------------|
| <i>instance</i>   | Instance number of the PWM module.       |
| <i>subModule</i>  | The FlexPWM submodule that is configured |
| <i>trigger</i>    | Trigger number that we wish to configure |
| <i>triggerVal</i> | Trigger value                            |

### 30.2.6.7 void PWM\_DRV\_SetupFault ( uint32\_t *instance*, pwm\_module\_t *subModule*, pwm\_fault\_input\_t *faultNum*, pwm\_fault\_setup\_t \* *faultParams*, bool *pwmA*, bool *pwmB*, bool *pwmX* )

This function configures a fault parameter and enables the fault for the appropriate sub-module signals.

Parameters

|                    |                                                                                   |
|--------------------|-----------------------------------------------------------------------------------|
| <i>instance</i>    | Instance number of the PWM module.                                                |
| <i>subModule</i>   | The FlexPWM submodule that is configured                                          |
| <i>faultNum</i>    | Fault that should be configured                                                   |
| <i>faultParams</i> | Parameters that configure the fault                                               |
| <i>pwmA</i>        | true: PWM A is disabled by this fault; false: PWM A is not affected by this fault |
| <i>pwmB</i>        | true: PWM B is disabled by this fault; false: PWM A is not affected by this fault |
| <i>pwmX</i>        | true: PWM X is disabled by this fault; false: PWM A is not affected by this fault |

**30.2.6.8 void PWM\_DRV\_CounterStart ( uint32\_t instance, uint8\_t value )**

This function starts the PWM submodule counters.

Parameters

|                 |                                                            |
|-----------------|------------------------------------------------------------|
| <i>instance</i> | Instance number of the PWM module.                         |
| <i>value</i>    | Submodules to start; 4 bit value, 1-bit for each submodule |

**30.2.6.9 void PWM\_DRV\_CounterStop ( uint32\_t instance, uint8\_t value )**

This function stops the the PWM submodule counters.

Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>instance</i> | Instance number of the PWM module.                        |
| <i>value</i>    | Submodules to stop; 4 bit value, 1-bit for each submodule |

**30.2.6.10 void PWM\_DRV\_SetExternalClkFreq ( uint32\_t instance, uint32\_t externalClkFreq )**

When using an external signal as clock source, the user should provide the frequency of this clock source so that this driver can calculate the register values used to generate the requested PWM signal.

Parameters

---

## eFlexPWM Peripheral Driver

|                         |                                    |
|-------------------------|------------------------------------|
| <i>instance</i>         | Instance number of the PWM module. |
| <i>externalClk-Freq</i> | External clock frequency (in Hz).  |

### 30.2.7 Variable Documentation

30.2.7.1 PWM\_Type\* const g\_pwmBase[PWM\_INSTANCE\_COUNT]

30.2.7.2 const IRQn\_Type g\_pwmCmplrqld[FSL\_FEATURE\_PWM\_CMP\_INT\_HANDLER\_COUNT]

## Chapter 31 FlexTimer (FTM)

### 31.1 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

### 31.2 FlexTimer HAL driver

#### 31.2.1 Overview

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

#### Data Structures

- struct `quadtmr_counter_params_t`  
*Quad timer counter parameters. [More...](#)*

#### Enumerations

- enum `quadtmr_pri_count_source_t`  
*Quad timer primary clock source selection.*
- enum `quadtmr_input_source_t`  
*Quad timer input sources selection.*
- enum `quadtmr_counting_mode_t` {  
`kQuadTmrCntNoOper = 0,`  
`kQuadTmrCntPriSrcRiseEdge,`  
`kQuadTmrCntPriSrcRiseAndFallEdge,`  
`kQuadTmrCntPriSrcRiseEdgeSecInpHigh,`  
`kQuadTmrQuadCntMode,`  
`kQuadTmrCntPriSrcRiseEdgeSecDir,`  
`kQuadTmrSecSrcTrigPriCnt,`  
`kQuadTmrCascadeCnt }`  
*Quad timer counting mode selection.*
- enum `quadtmr_output_mode_t` {  
`kQuadTmrAssertWhenCntActive = 0,`  
`kQuadTmrClearOnCompare,`  
`kQuadTmrSetOnCompare,`  
`kQuadTmrTogOnCompare,`  
`kQuadTmrTogOnAltCompReg,`  
`kQuadTmrSetOnCompareClearOnSecSrcInp,`  
`kQuadTmrSetOnCompareClearOnCntRoll,`  
`kQuadTmrEnableGateClk }`  
*Quad timer output mode selection.*
- enum `quadtmr_input_capture_edge_mode_t` {  
`kQuadTmrNoCapture = 0,`  
`kQuadTmrRisingEdge,`  
`kQuadTmrFallingEdge,`  
`kQuadTmrRisingAndFallingEdge }`  
*Quad timers input capture edge mode, rising edge, or falling edge.*
- enum `quadtmr_preload_control_t` {



```
kQuadTmrNoPreload = 0,
kQuadTmrLoadOnComp1,
kQuadTmrLoadOnComp2 }
```

*Quad timers input capture edge mode, rising edge, or falling edge.*

## Functions

- void [QUADTMR\\_HAL\\_Init](#) (TMR\_Type \*tmrBase)  
*Initializes the Quad Timer registers.*
- void [QUADTMR\\_HAL\\_SetupCounter](#) (TMR\_Type \*tmrBase, [quadtmr\\_counter\\_params\\_t](#) \*params, bool src\_polarity)  
*Sets the Quad timer's counter properties.*
- void [QUADTMR\\_HAL\\_ForceOutVal](#) (TMR\_Type \*tmrBase, bool val, bool pol)  
*Enables the user to write a value out on the OFLAG pin.*
- void [QUADTMR\\_HAL\\_OutPulses](#) (TMR\_Type \*tmrBase, uint16\_t numOfPulses, [quadtmr\\_pri\\_count\\_source\\_t](#) src)  
*Enables the user to output a certain number of pulses.*
- void [QUADTMR\\_HAL\\_OutPwmSignal](#) (TMR\_Type \*tmrBase, [quadtmr\\_pri\\_count\\_source\\_t](#) src, uint16\_t highPulseCount, uint16\_t lowPulseCount)  
*Outputs a PWM signal on the OFLAG pin.*
- void [QUADTMR\\_HAL\\_SetupInputCapture](#) (TMR\_Type \*tmrBase, [quadtmr\\_pri\\_count\\_source\\_t](#) countSrc, [quadtmr\\_input\\_source\\_t](#) captureSrc, [quadtmr\\_input\\_capture\\_edge\\_mode\\_t](#) mode, bool inputPol)  
*Allows the user to count the source clock cycles until a capture event arrives.*
- void [QUADTMR\\_HAL\\_SetupComparePreload](#) (TMR\_Type \*tmrBase, uint8\_t compareRegNum, uint16\_t preloadVal, [quadtmr\\_preload\\_control\\_t](#) preloadProps)  
*Sets up the Quad timers compare preload feature.*
- static void [QUADTMR\\_HAL\\_SetCounter](#) (TMR\_Type \*tmrBase, uint16\_t val)  
*Sets the Quad timer current counter value.*
- static void [QUADTMR\\_HAL\\_SetComp1Val](#) (TMR\_Type \*tmrBase, uint16\_t val)  
*Sets the Quad Timer compare register 1 value used in count up mode.*
- static void [QUADTMR\\_HAL\\_SetComp1PreloadVal](#) (TMR\_Type \*tmrBase, uint16\_t val)  
*Sets up the preload register for Quad Timer compare 1 register.*
- static void [QUADTMR\\_HAL\\_SetComp2Val](#) (TMR\_Type \*tmrBase, uint16\_t val)  
*Sets the Quad Timer compare register 2 value used in count up mode.*
- static void [QUADTMR\\_HAL\\_SetComp2PreloadVal](#) (TMR\_Type \*tmrBase, uint16\_t val)  
*Sets up the preload register for Quad Timer compare 2 register.*
- static void [QUADTMR\\_HAL\\_SetLoadVal](#) (TMR\_Type \*tmrBase, uint16\_t val)  
*Sets the Quad timer counter initial value.*
- static uint16\_t [QUADTMR\\_HAL\\_GetHoldVal](#) (TMR\_Type \*tmrBase)  
*Returns the Quad timer peripheral hold register value.*
- static uint16\_t [QUADTMR\\_HAL\\_GetCaptureVal](#) (TMR\_Type \*tmrBase)  
*Gets the Quad timer peripherals counter capture value.*
- static uint16\_t [QUADTMR\\_HAL\\_GetCounterVal](#) (TMR\_Type \*tmrBase)  
*Gets the Quad timer peripherals counter value.*
- static void [QUADTMR\\_HAL\\_SetCountMode](#) (TMR\_Type \*tmrBase, [quadtmr\\_counting\\_mode\\_t](#) mode)  
*Sets up the Quad timer counter mode.*

## FlexTimer HAL driver

- static void [QUADTMR\\_HAL\\_SetPriCountSrc](#) (TMR\_Type \*tmrBase, [quadtmr\\_pri\\_count\\_source\\_t](#) src)  
*Sets up the Quad timer primary count source.*
- static void [QUADTMR\\_HAL\\_SetSecCountSrc](#) (TMR\_Type \*tmrBase, [quadtmr\\_input\\_source\\_t](#) src)  
*Sets up the Quad timer secondary count source.*
- static void [QUADTMR\\_HAL\\_SetCountOnceCmd](#) (TMR\_Type \*tmrBase, bool once)  
*Sets up the count once bit.*
- static void [QUADTMR\\_HAL\\_SetOutputMode](#) (TMR\_Type \*tmrBase, [quadtmr\\_output\\_mode\\_t](#) mode)  
*Sets up the Quad timer output mode.*
- static void [QUADTMR\\_HAL\\_SetInputPolSelCmd](#) (TMR\_Type \*tmrBase, bool ips)  
*Modifies the Quad timers input polarity select bit.*
- static bool [QUADTMR\\_HAL\\_GetExtInputSignal](#) (TMR\_Type \*tmrBase)  
*Reads the Quad timers external input signal bit.*
- static void [QUADTMR\\_HAL\\_SetCaptureMode](#) (TMR\_Type \*tmrBase, [quadtmr\\_input\\_capture\\_edge\\_mode\\_t](#) mode)  
*Sets the Quad timers input capture mode.*
- static void [QUADTMR\\_HAL\\_SetOutEnableCmd](#) (TMR\_Type \*tmrBase, bool enable)  
*Modifies the value for the Quad timers output enable bit.*
- static void [QUADTMR\\_HAL\\_SetInputFilterCnt](#) (TMR\_Type \*tmrBase, uint8\_t cnt)  
*Sets input filter sample count.*
- static void [QUADTMR\\_HAL\\_SetInputFilterPeriod](#) (TMR\_Type \*tmrBase, uint8\_t period)  
*Sets input filter sample period.*

## 31.2.2 Data Structure Documentation

### 31.2.2.1 struct [quadtmr\\_counter\\_params\\_t](#)

#### Data Fields

- uint16\_t [counterVal](#)  
*Value to be written to the counter register.*
- uint16\_t [counterLoadVal](#)  
*Value to be loaded into the counter on compare or overflow.*
- uint16\_t [countUpCompareVal](#)  
*Value to be written to the count up compare register.*
- uint16\_t [countDownCompareVal](#)  
*Value to be written to the count down compare register.*
- [quadtmr\\_pri\\_count\\_source\\_t](#) [primarySrc](#)  
*Specify the primary count source.*
- [quadtmr\\_input\\_source\\_t](#) [secondarySrc](#)  
*Specify the secondary count source.*
- bool [countOnce](#)  
*true: if counter should stop on compare, false: to run repeatedly*
- bool [countLength](#)  
*true: re-initialize on compare match, false: count till roll-over*
- bool [countDir](#)  
*true: Count down, false: Count up*

### 31.2.3 Enumeration Type Documentation

#### 31.2.3.1 enum quadtmr\_counting\_mode\_t

Enumerator

*kQuadTmrCntNoOper* No operation.  
*kQuadTmrCntPriSrcRiseEdge* Count rising edges or primary source.  
*kQuadTmrCntPriSrcRiseAndFallEdge* Count rising and falling edges of primary source.  
*kQuadTmrCntPriSrcRiseEdgeSecInpHigh* Count rise edges of pri src while sec inp high active.  
*kQuadTmrQuadCntMode* Quadature count mode, uses pri and sec sources.  
*kQuadTmrCntPriSrcRiseEdgeSecDir* Count rising edges of pri src; sec src specifies dir.  
*kQuadTmrSecSrcTrigPriCnt* Edge of sec src trigger primary count until compare.  
*kQuadTmrCascadeCnt* Cascaded count mode (up/down)

#### 31.2.3.2 enum quadtmr\_output\_mode\_t

Enumerator

*kQuadTmrAssertWhenCntActive* Assert OFLAG while counter is active.  
*kQuadTmrClearOnCompare* Clear OFLAG on successful compare.  
*kQuadTmrSetOnCompare* Set OFLAG on successful compare.  
*kQuadTmrTogOnCompare* Toggle OFLAG on successful compare.  
*kQuadTmrTogOnAltCompReg* Toggle OFLAG using alternating compare registers.  
*kQuadTmrSetOnCompareClearOnSecSrcInp* Set OFLAG on compare, clear on sec src input edge.  
  
*kQuadTmrSetOnCompareClearOnCntRoll* Set OFLAG on compare, clear on counter rollover.  
*kQuadTmrEnableGateClk* Enable gated clock output while count is active.

#### 31.2.3.3 enum quadtmr\_input\_capture\_edge\_mode\_t

Enumerator

*kQuadTmrNoCapture* Capture is disabled.  
*kQuadTmrRisingEdge* Capture on rising edge (IPS=0) or falling edge (IPS=1)  
*kQuadTmrFallingEdge* Capture on falling edge (IPS=0) or rising edge (IPS=1)  
*kQuadTmrRisingAndFallingEdge* Capture on both edges.

#### 31.2.3.4 enum quadtmr\_preload\_control\_t

Enumerator

*kQuadTmrNoPreload* Never preload.

## FlexTimer HAL driver

*kQuadTmrLoadOnComp1* Load upon successful compare with value in COMP1.

*kQuadTmrLoadOnComp2* Load upon successful compare with value in COMP2.

### 31.2.4 Function Documentation

#### 31.2.4.1 void QUADTMR\_HAL\_Init ( TMR\_Type \* *tmrBase* )

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer |
|----------------|-------------------------------------|

#### 31.2.4.2 void QUADTMR\_HAL\_SetupCounter ( TMR\_Type \* *tmrBase*, quadtmr\_counter\_params\_t \* *params*, bool *src\_polarity* )

The primary source and in some cases secondary clock control how the to increment the counter. The parameters provided by the user will be used to setup counter properties like match value, rollover behaviour, direction. The function will not start the counter, the user will have to make a separate call to QUADTMR\_HAL\_SetCountMode to set count mode.

Parameters

|                     |                                                                              |
|---------------------|------------------------------------------------------------------------------|
| <i>tmrBase</i>      | The Quad timer base address pointer                                          |
| <i>params</i>       | Used to setup the counter behaviour.                                         |
| <i>src_polarity</i> | true: Source clock will be inverted, false: no inversion of the input signal |

#### 31.2.4.3 void QUADTMR\_HAL\_ForceOutVal ( TMR\_Type \* *tmrBase*, bool *val*, bool *pol* )

Parameters

|                |                                                                 |
|----------------|-----------------------------------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer                             |
| <i>val</i>     | true: high value is written, false: low value is written        |
| <i>pol</i>     | true: invert polarity of the output signal, false: no inversion |

#### 31.2.4.4 void QUADTMR\_HAL\_OutPulses ( TMR\_Type \* *tmrBase*, uint16\_t *numOfPulses*, quadtmr\_pri\_count\_source\_t *src* )

The number of pulses to be outputted is provided as an argument. The output pulse stream has the same frequency as the source clock provided.

## Parameters

|                    |                                                                        |
|--------------------|------------------------------------------------------------------------|
| <i>tmrBase</i>     | The Quad timer base address pointer                                    |
| <i>numOfPulses</i> | The number of pulses to be outputted on the OFLAG pin                  |
| <i>src</i>         | Source clock for the counter that is used to generate the pulse stream |

**31.2.4.5 void QUADTMR\_HAL\_OutPwmSignal ( TMR\_Type \* *tmrBase*,  
quadtmr\_pri\_count\_source\_t *src*, uint16\_t *highPulseCount*, uint16\_t  
*lowPulseCount* )**

The high and low pulse width values should be a count in terms of number of source clock cycles. The user has to calculate these values based on the source clock frequency, desired PWM frequency and duty cycle.

## Parameters

|                        |                                                                            |
|------------------------|----------------------------------------------------------------------------|
| <i>tmrBase</i>         | The Quad timer base address pointer                                        |
| <i>src</i>             | Source clock for the counter that is used to generate the PWM signal       |
| <i>highPulse-Count</i> | Specify the number of source clock cycles to get the high period           |
| <i>low</i>             | PulseCount Specify the number of source clock cycles to get the low period |

**31.2.4.6 void QUADTMR\_HAL\_SetupInputCapture ( TMR\_Type \* *tmrBase*,  
quadtmr\_pri\_count\_source\_t *countSrc*, quadtmr\_input\_source\_t *captureSrc*,  
quadtmr\_input\_capture\_edge\_mode\_t *mode*, bool *inputPol* )**

The count will be stored in the capture register.

## Parameters

|                   |                                                                           |
|-------------------|---------------------------------------------------------------------------|
| <i>tmrBase</i>    | The Quad timer base address pointer                                       |
| <i>src</i>        | Source clock for the counter                                              |
| <i>captureSrc</i> | Pin through which we will receive the input signal to trigger the capture |
| <i>mode</i>       | Specifies which edge of the input signal will trigger a capture           |

## FlexTimer HAL driver

|            |                                                                |
|------------|----------------------------------------------------------------|
| <i>pol</i> | true: invert polarity of the input signal, false: no inversion |
|------------|----------------------------------------------------------------|

**31.2.4.7** void QUADTMR\_HAL\_SetupComparePreload ( TMR\_Type \* *tmrBase*, uint8\_t *compareRegNum*, uint16\_t *preloadVal*, quadtmr\_preload\_control\_t *preloadProps* )

### Parameters

|                       |                                                                                    |
|-----------------------|------------------------------------------------------------------------------------|
| <i>tmrBase</i>        | The Quad timer base address pointer                                                |
| <i>compareReg-Num</i> | Options are 1 or 2, 1=COMP1; 2=COMP2                                               |
| <i>preloadVal</i>     | Value to be written to the comparators preload register                            |
| <i>preloadProps</i>   | Controls when a compare register is preloaded with the value from the preload reg. |

**31.2.4.8** static void QUADTMR\_HAL\_SetCounter ( TMR\_Type \* *tmrBase*, uint16\_t *val* )  
[inline], [static]

### Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer |
| <i>val</i>     | Quad timer counter value to be set  |

**31.2.4.9** static void QUADTMR\_HAL\_SetComp1Val ( TMR\_Type \* *tmrBase*, uint16\_t *val* )  
[inline], [static]

### Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>tmrBase</i> | The Quad Timer base address pointer |
| <i>val</i>     | The value to be set                 |

**31.2.4.10** static void QUADTMR\_HAL\_SetComp1PreloadVal ( TMR\_Type \* *tmrBase*, uint16\_t *val* ) [inline], [static]

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>tmrBase</i> | The Quad Timer base address pointer |
| <i>val</i>     | The value to be set                 |

**31.2.4.11** `static void QUADTMR_HAL_SetComp2Val ( TMR_Type * tmrBase, uint16_t val ) [inline], [static]`

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>tmrBase</i> | The Quad Timer base address pointer |
| <i>val</i>     | The value to be set                 |

**31.2.4.12** `static void QUADTMR_HAL_SetComp2PreloadVal ( TMR_Type * tmrBase, uint16_t val ) [inline], [static]`

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>tmrBase</i> | The Quad Timer base address pointer |
| <i>val</i>     | The value to be set                 |

**31.2.4.13** `static void QUADTMR_HAL_SetLoadVal ( TMR_Type * tmrBase, uint16_t val ) [inline], [static]`

Parameters

|                |                              |
|----------------|------------------------------|
| <i>tmrBase</i> | The FTM base address pointer |
| <i>val</i>     | initial value to be set      |

**31.2.4.14** `static uint16_t QUADTMR_HAL_GetHoldVal ( TMR_Type * tmrBase ) [inline], [static]`

Parameters

---

## FlexTimer HAL driver

|                |                                     |
|----------------|-------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer |
|----------------|-------------------------------------|

Returns

Quad timer hold value

**31.2.4.15** `static uint16_t QUADTMR_HAL_GetCaptureVal ( TMR_Type * tmrBase )  
[inline], [static]`

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer |
|----------------|-------------------------------------|

Returns

Returns the value captured from the counter

**31.2.4.16** `static uint16_t QUADTMR_HAL_GetCounterVal ( TMR_Type * tmrBase )  
[inline], [static]`

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer |
|----------------|-------------------------------------|

Returns

Returns the counter value

**31.2.4.17** `static void QUADTMR_HAL_SetCountMode ( TMR_Type * tmrBase,  
quadtmr_counting_mode_t mode ) [inline], [static]`

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer |
|----------------|-------------------------------------|



|             |                                                                            |
|-------------|----------------------------------------------------------------------------|
| <i>mode</i> | Count mode, options available in <code>quadtmr_counting_mode_t</code> enum |
|-------------|----------------------------------------------------------------------------|

**31.2.4.18** `static void QUADTMR_HAL_SetPriCountSrc ( TMR_Type * tmrBase, quadtmr_pri_count_source_t src ) [inline], [static]`

Parameters

|                |                                                                                             |
|----------------|---------------------------------------------------------------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer                                                         |
| <i>src</i>     | Primary count source, options available in the <code>quadtmr_pri_count_source_t</code> enum |

**31.2.4.19** `static void QUADTMR_HAL_SetSecCountSrc ( TMR_Type * tmrBase, quadtmr_input_source_t src ) [inline], [static]`

Parameters

|                |                                                                                           |
|----------------|-------------------------------------------------------------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer                                                       |
| <i>src</i>     | Secondary count source, options available in the <code>quadtmr_input_source_t</code> enum |

**31.2.4.20** `static void QUADTMR_HAL_SetCountOnceCmd ( TMR_Type * tmrBase, bool once ) [inline], [static]`

Parameters

|                |                                                                  |
|----------------|------------------------------------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer                              |
| <i>once</i>    | true: count until compare and then stop, false: count repeatedly |

**31.2.4.21** `static void QUADTMR_HAL_SetOutputMode ( TMR_Type * tmrBase, quadtmr_output_mode_t mode ) [inline], [static]`

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer |
|----------------|-------------------------------------|

## FlexTimer HAL driver

|             |                                                                               |
|-------------|-------------------------------------------------------------------------------|
| <i>mode</i> | Output mode, options available in the <code>quadtmr_output_mode_t</code> enum |
|-------------|-------------------------------------------------------------------------------|

**31.2.4.22** `static void QUADTMR_HAL_SetInputPolSelCmd ( TMR_Type * tmrBase, bool ips ) [inline], [static]`

### Parameters

|                |                                                              |
|----------------|--------------------------------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer                          |
| <i>ips</i>     | true: inverts the input signal polarity, false: no inversion |

**31.2.4.23** `static bool QUADTMR_HAL_GetExtInputSignal ( TMR_Type * tmrBase ) [inline], [static]`

### Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer |
|----------------|-------------------------------------|

### Returns

Returns the current state of the external input pin selected via the sec. count source

**31.2.4.24** `static void QUADTMR_HAL_SetCaptureMode ( TMR_Type * tmrBase, quadtmr_input_capture_edge_mode_t mode ) [inline], [static]`

### Parameters

|                |                                                                                            |
|----------------|--------------------------------------------------------------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer                                                        |
| <i>mode</i>    | Capture mode, options specified in the <code>quadtmr_input_capture_edge_mode_t</code> enum |

**31.2.4.25** `static void QUADTMR_HAL_SetOutEnableCmd ( TMR_Type * tmrBase, bool enable ) [inline], [static]`

Parameters

|                |                                                                                  |
|----------------|----------------------------------------------------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer                                              |
| <i>enable</i>  | true: OFLAG output signal driven on pin, false: External pin configured as input |

**31.2.4.26 static void QUADTMR\_HAL\_SetInputFilterCnt ( TMR\_Type \* *tmrBase*, uint8\_t *cnt* ) [inline], [static]**

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer |
| <i>count</i>   | Input filter sample count           |

**31.2.4.27 static void QUADTMR\_HAL\_SetInputFilterPeriod ( TMR\_Type \* *tmrBase*, uint8\_t *period* ) [inline], [static]**

Parameters

|                |                                     |
|----------------|-------------------------------------|
| <i>tmrBase</i> | The Quad timer base address pointer |
| <i>count</i>   | Input filter sample period          |

## FlexTimer Peripheral Driver

### 31.3 FlexTimer Peripheral Driver

#### 31.3.1 Overview

This section describes the programming interface of the FlexTimer Peripheral driver.

#### 31.3.2 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.

#### 31.3.3 FlexTimer Initialization

1. To initialize the FlexTimer driver, call the `FTM_DRV_Init()` function and pass the instance number of the relevant FTM. 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 tofFrequency;
 bool isFTMMode;
 uint8_t BDMMode;
 bool isWriteProtection;

 bool isTimerOverFlowInterrupt;
 bool isFaultInterrupt;
} ftm_user_config_t;
```

#### 31.3.4 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;

 uint16_t uFirstEdgeDelayPercent;
} ftm_pwm_param_t;
```

0 = inactive signal (0% duty cycle)...  
100 = active signal (100% duty cycle). //

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//

The mode options are `kFtmEdgeAlignedPWM`, `kFtmCenterAlignedPWM`, and `kFtmCombinedPWM`. For edge mode, the options available are `kFtmHighTrue` and `kFtmLowTrue`. Specify the PWM signal frequency in Hertz 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 is delayed.

### Enumerations

- enum `quadtmr_ip_bus_clock_source_t` {  
`kQuadTmrIpBusClkDiv1 = 0,`  
`kQuadTmrIpBusClkDiv2,`  
`kQuadTmrIpBusClkDiv4,`  
`kQuadTmrIpBusClkDiv8,`  
`kQuadTmrIpBusClkDiv16,`  
`kQuadTmrIpBusClkDiv32,`  
`kQuadTmrIpBusClkDiv64,`  
`kQuadTmrIpBusClkDiv128 }`  
*Quad timer IP bus clock options to run the counter.*

### Functions

- void `QUADTMR_DRV_Init` (`uint8_t` instance)  
*Initializes the Quad Timer driver.*
- void `QUADTMR_DRV_Deinit` (`uint8_t` instance)  
*Shuts down the Quad Timer driver.*
- void `QUADTMR_DRV_Start64BitCounter` (`quadtmr_pri_count_source_t` clk)  
*Provides a 64-bit counter.*
- `uint64_t` `QUADTMR_DRV_Get64BitCountVal` (void)  
*Gets the current count value when running in 64-bit mode.*
- void `QUADTMR_DRV_SetupFlexPwm` (`uint8_t` instance, `quadtmr_ip_bus_clock_source_t` clockSrc, `uint32_t` pwmPulsePeriod, `uint32_t` pulseWidthPeriod)  
*Provides the PWM signal.*
- void `QUADTMR_DRV_UpdatePwm` (`uint8_t` instance, `uint32_t` pwmPulsePeriod, `uint32_t` pulseWidthPeriod)  
*Provides a way to update the current PWM signal.*
- void `QUADTMR_DRV_IRQHandler` (`uint8_t` instance)  
*Action to take when a Quad Timer interrupt is triggered.*

### Variables

- `TMR_Type` \*const `g_quadtmrBase` [`TMR_INSTANCE_COUNT`]  
*Table of base addresses for Quad Timer instances.*
- const `IRQn_Type` `g_quadtmrIrqId` [`TMR_INSTANCE_COUNT`]  
*Table to save Quad Timer IRQ enumeration numbers defined in the CMSIS header file.*

## FlexTimer Peripheral Driver

### 31.3.5 Enumeration Type Documentation

#### 31.3.5.1 enum quadtmr\_ip\_bus\_clock\_source\_t

Enumerator

*kQuadTmrIpBusClkDiv1* IP Bus clock Div 1.  
*kQuadTmrIpBusClkDiv2* IP Bus clock Div 2.  
*kQuadTmrIpBusClkDiv4* IP Bus clock Div 4.  
*kQuadTmrIpBusClkDiv8* IP Bus clock Div 8.  
*kQuadTmrIpBusClkDiv16* IP Bus clock Div 16.  
*kQuadTmrIpBusClkDiv32* IP Bus clock Div 32.  
*kQuadTmrIpBusClkDiv64* IP Bus clock Div 64.  
*kQuadTmrIpBusClkDiv128* IP Bus clock Div 128.

### 31.3.6 Function Documentation

#### 31.3.6.1 void QUADTMR\_DRV\_Init ( uint8\_t instance )

Initializes the Quad Timer registers, un-gates the module clock and enables Quad Timer interrupt in the system interrupt controller.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The Quad Timer peripheral instance number. |
|-----------------|--------------------------------------------|

#### 31.3.6.2 void QUADTMR\_DRV\_Deinit ( uint8\_t instance )

Gates the module clock and disables the interrupt in the system interrupt controller.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The Quad Timer peripheral instance number. |
|-----------------|--------------------------------------------|

#### 31.3.6.3 void QUADTMR\_DRV\_Start64BitCounter ( quadtmr\_pri\_count\_source\_t clk )

The user gets to choose which source clock is used by the counter. The user must ensure that the [QUADTMR\\_DRV\\_Init\(\)](#) function is called on all timer instances because all timer instances are used in this mode. Timer 0 is clocked via the source clock and Timers 1, 2, and 3 are set up in the cascade mode. All 4 counters count up to 0xFFFF, then roll-over and continue counting.

### Parameters

|            |                                          |
|------------|------------------------------------------|
| <i>clk</i> | Source clock that feeds timer 0 counter. |
|------------|------------------------------------------|

#### 31.3.6.4 uint64\_t QUADTMR\_DRV\_Get64BitCountVal ( void )

The function reads all 4 timer values and returns the value.

### Returns

The current value of all 4 counters is concatenated to a 64-bit value.

#### 31.3.6.5 void QUADTMR\_DRV\_SetupFlexPwm ( uint8\_t instance, quadtmr\_ip\_bus\_clock\_source\_t clockSrc, uint32\_t pwmPulsePeriod, uint32\_t pulseWidthPeriod )

The user provides the input clock source which is derived from the IP Bus clock. The user also provides the desired PWM signal's period in microseconds and the pulse width in microseconds. This function enables the pre-load function for each compare register to allow updating the PWM signal characteristics at a later time.

### Parameters

|                          |                                                                               |
|--------------------------|-------------------------------------------------------------------------------|
| <i>instance</i>          | The Quad timer peripheral instance number.                                    |
| <i>clockSrc</i>          | Counter source clock, options available in quadtmr_ip_bus_clock_source_t enum |
| <i>pwmPulse-Period</i>   | PWM period specified in microseconds                                          |
| <i>pulseWidth-Period</i> | PWM pulse width specified in microseconds                                     |

#### 31.3.6.6 void QUADTMR\_DRV\_UpdatePwm ( uint8\_t instance, uint32\_t pwmPulsePeriod, uint32\_t pulseWidthPeriod )

Prior to calling this function, the user should call the [QUADTMR\\_DRV\\_SetupFlexPwm\(\)](#) function to start outputting a PWM signal. Call this function to update the PWM's period and/or pulse width. The PWM continues to be derived from the clock source provided during setup.

## FlexTimer Peripheral Driver

### Parameters

|                          |                                                   |
|--------------------------|---------------------------------------------------|
| <i>instance</i>          | The Quad timer peripheral instance number.        |
| <i>pwmPulse-Period</i>   | Updated PWM period specified in microseconds      |
| <i>pulseWidth-Period</i> | Updated PWM pulse width specified in microseconds |

### 31.3.6.7 void QUADTMR\_DRV\_IRQHandler ( uint8\_t *instance* )

The timer compare flags are checked and cleared if set.

### Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The Quad timer peripheral instance number. |
|-----------------|--------------------------------------------|

## 31.3.7 Variable Documentation

31.3.7.1 TMR\_Type\* const g\_quadtmrBase[TMR\_INSTANCE\_COUNT]

31.3.7.2 const IRQn\_Type g\_quadtmrIrqId[TMR\_INSTANCE\_COUNT]



---

## Chapter 32

# Random Number Generator Accelerator (RNGA)

### 32.1 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

### 32.2 RNGA HAL driver

#### 32.2.1 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` (RNG\_Type \*base)  
*Initializes the RNGA module.*
- static void `RNGA_HAL_Enable` (RNG\_Type \*base)  
*Enables the RNGA module.*
- static void `RNGA_HAL_Disable` (RNG\_Type \*base)  
*Disables the RNGA module.*
- static void `RNGA_HAL_SetHighAssuranceCmd` (RNG\_Type \*base, bool enable)  
*Sets the RNGA high assurance.*
- static void `RNGA_HAL_SetIntMaskCmd` (RNG\_Type \*base, bool enable)  
*Sets the RNGA interrupt mask.*
- static void `RNGA_HAL_ClearIntFlag` (RNG\_Type \*base, bool enable)  
*Clears the RNGA interrupt.*
- static void `RNGA_HAL_SetWorkModeCmd` (RNG\_Type \*base, `rnga_mode_t` mode)  
*Sets the RNGA in sleep mode or normal mode.*
- static uint8\_t `RNGA_HAL_GetOutputRegSize` (RNG\_Type \*base)  
*Gets the output register size.*

- static `rnga_output_reg_level_t` `RNGA_HAL_GetOutputRegLevel` (`RNG_Type *base`)  
*Gets the output register level.*
- static `rnga_mode_t` `RNGA_HAL_GetWorkMode` (`RNG_Type *base`)  
*Gets the RNGA working mode.*
- static `bool` `RNGA_HAL_GetErrorIntCmd` (`RNG_Type *base`)  
*Gets the RNGA status whether an error interrupt has occurred.*
- static `bool` `RNGA_HAL_GetOutputRegUnderflowCmd` (`RNG_Type *base`)  
*Gets the RNGA status whether an output register underflow has occurred.*
- static `bool` `RNGA_HAL_GetLastReadStatusCmd` (`RNG_Type *base`)  
*Gets the most recent RNGA read status.*
- static `bool` `RNGA_HAL_GetSecurityViolationCmd` (`RNG_Type *base`)  
*Gets the RNGA status whether a security violation has occurred.*
- static `uint32_t` `RNGA_HAL_ReadRandomData` (`RNG_Type *base`)  
*Gets a random data from the RNGA.*
- `rnga_status_t` `RNGA_HAL_GetRandomData` (`RNG_Type *base`, `uint32_t *data`)  
*Get random data.*
- static `void` `RNGA_HAL_WriteSeed` (`RNG_Type *base`, `uint32_t data`)  
*Inputs an entropy value used to seed the RNGA.*

## 32.2.2 Enumeration Type Documentation

### 32.2.2.1 enum `rnga_mode_t`

Enumerator

- `kRNGAModeNormal` Normal Mode.
- `kRNGAModeSleep` Sleep Mode.

### 32.2.2.2 enum `rnga_output_reg_level_t`

Enumerator

- `kRNGAOutputRegLevelNowords` output register no words.
- `kRNGAOutputRegLevelOneword` output register one word.

### 32.2.2.3 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.

## RNGA HAL driver

### 32.2.3 Function Documentation

#### 32.2.3.1 `static void RNGA_HAL_Init ( RNG_Type * base ) [inline], [static]`

This function initializes the RNGA to a default state.

Parameters

|                  |              |
|------------------|--------------|
| <i>base,RNGA</i> | base address |
|------------------|--------------|

### 32.2.3.2 static void RNGA\_HAL\_Enable ( RNG\_Type \* *base* ) [inline], [static]

This function enables the RNGA random data generation and loading.

Parameters

|                  |              |
|------------------|--------------|
| <i>base,RNGA</i> | base address |
|------------------|--------------|

### 32.2.3.3 static void RNGA\_HAL\_Disable ( RNG\_Type \* *base* ) [inline], [static]

This function disables the RNGA module.

Parameters

|                  |              |
|------------------|--------------|
| <i>base,RNGA</i> | base address |
|------------------|--------------|

### 32.2.3.4 static void RNGA\_HAL\_SetHighAssuranceCmd ( RNG\_Type \* *base*, bool *enable* ) [inline], [static]

This function sets the RNGA high assurance(notification of security violations).

Parameters

|                  |                                                                                                          |
|------------------|----------------------------------------------------------------------------------------------------------|
| <i>base,RNGA</i> | base address                                                                                             |
| <i>enable,0</i>  | means notification of security violations disabled. 1 means notification of security violations enabled. |

### 32.2.3.5 static void RNGA\_HAL\_SetIntMaskCmd ( RNG\_Type \* *base*, bool *enable* ) [inline], [static]

This function sets the RNGA error interrupt mask.

## RNGA HAL driver

Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base,RNGA</i> | base address                                              |
| <i>enable,0</i>  | means unmask RNGA interrupt. 1 means mask RNGA interrupt. |

### 32.2.3.6 static void RNGA\_HAL\_ClearIntFlag ( RNG\_Type \* *base*, bool *enable* ) [inline], [static]

This function clears the RNGA interrupt.

Parameters

|                  |                                                                |
|------------------|----------------------------------------------------------------|
| <i>base,RNGA</i> | base address                                                   |
| <i>enable,0</i>  | means do not clear the interrupt. 1 means clear the interrupt. |

### 32.2.3.7 static void RNGA\_HAL\_SetWorkModeCmd ( RNG\_Type \* *base*, rnga\_mode\_t *mode* ) [inline], [static]

This function specifies whether the RNGA is in sleep mode or normal mode.

Parameters

|                              |                                                                             |
|------------------------------|-----------------------------------------------------------------------------|
| <i>base,RNGA</i>             | base address                                                                |
| <i>mode,kRNGA-ModeNormal</i> | means set RNGA in normal mode. kRNGAModeSleep means set RNGA in sleep mode. |

### 32.2.3.8 static uint8\_t RNGA\_HAL\_GetOutputRegSize ( RNG\_Type \* *base* ) [inline], [static]

This function gets the size of the output register as 32-bit random data words it can hold.

Parameters

|                  |              |
|------------------|--------------|
| <i>base,RNGA</i> | base address |
|------------------|--------------|

Returns

1 means one word(this value is fixed).

**32.2.3.9** `static rnga_output_reg_level_t RNGA_HAL_GetOutputRegLevel ( RNG_Type *  
base ) [inline], [static]`

This function gets the number of random-data words that are in OR [RANDOUT], which indicates if OR is valid.

## RNGA HAL driver

Parameters

|                  |              |
|------------------|--------------|
| <i>base,RNGA</i> | base address |
|------------------|--------------|

Returns

0 means no words(empty), 1 means one word(valid).

### 32.2.3.10 **static rnga\_mode\_t RNGA\_HAL\_GetWorkMode ( RNG\_Type \* *base* ) [inline], [static]**

This function checks whether the RNGA works in sleep mode or normal mode.

Parameters

|                  |              |
|------------------|--------------|
| <i>base,RNGA</i> | base address |
|------------------|--------------|

Returns

Kmode\_RNGA\_Normal means in normal mode Kmode\_RNGA\_Sleep means in sleep mode

### 32.2.3.11 **static bool RNGA\_HAL\_GetErrorIntCmd ( RNG\_Type \* *base* ) [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.

Parameters

|                  |              |
|------------------|--------------|
| <i>base,RNGA</i> | base address |
|------------------|--------------|

Returns

0 means no underflow, 1 means underflow

### 32.2.3.12 **static bool RNGA\_HAL\_GetOutputRegUnderflowCmd ( RNG\_Type \* *base* ) [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>base,RNGA</i> | base address |
|------------------|--------------|

Returns

0 means no underflow, 1 means underflow

**32.2.3.13 static bool RNGA\_HAL\_GetLastReadStatusCmd ( RNG\_Type \* *base* )  
[inline], [static]**

This function gets the RNGA status whether the most recent read of OR[RANDOUT] causes an OR underflow condition.

Parameters

|                  |              |
|------------------|--------------|
| <i>base,RNGA</i> | base address |
|------------------|--------------|

Returns

0 means no underflow, 1 means underflow

**32.2.3.14 static bool RNGA\_HAL\_GetSecurityViolationCmd ( RNG\_Type \* *base* )  
[inline], [static]**

This function gets the RNGA status whether a security violation has occurred when high assurance is enabled.

Parameters

|                  |              |
|------------------|--------------|
| <i>base,RNGA</i> | base address |
|------------------|--------------|

Returns

0 means no security violation, 1 means security violation

**32.2.3.15 static uint32\_t RNGA\_HAL\_ReadRandomData ( RNG\_Type \* *base* )  
[inline], [static]**

This function gets a random data from RNGA.

## RNGA HAL driver

Parameters

|                  |              |
|------------------|--------------|
| <i>base,RNGA</i> | base address |
|------------------|--------------|

Returns

random data obtained

### 32.2.3.16 `rnga_status_t RNGA_HAL_GetRandomData ( RNG_Type * base, uint32_t * data )`

This function is used to get a random data from RNGA

Parameters

|                     |                                   |
|---------------------|-----------------------------------|
| <i>base,RNGA</i>    | base address                      |
| <i>data,pointer</i> | address used to store random data |

Returns

one random data

### 32.2.3.17 `static void RNGA_HAL_WriteSeed ( RNG_Type * base, uint32_t data )` `[inline], [static]`

This function specifies an entropy value that RNGA uses with its ring oscillations to seed its pseudorandom algorithm.

Parameters

|                      |               |
|----------------------|---------------|
| <i>base,RNGA</i>     | base address  |
| <i>data,external</i> | entropy value |

## 32.3 RNGA Peripheral Driver

### 32.3.1 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.

### 32.3.2 RNGA Initialization

1. To initialize the RNGA module, call the [RNGA\\_DRV\\_Init\(\)](#) function and pass 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);
```

### 32.3.3 RNGA Set/Get Working Mode

The RNGA works either in sleep mode or normal mode

1. [RNGA\\_DRV\\_SetMode\(\)](#) function sets the RNGA mode.
2. [RNGA\\_DRV\\_GetMode\(\)](#) function gets the RNGA working mode.

### 32.3.4 Get random data from RNGA

1. [RNGA\\_DRV\\_GetRandomData\(\)](#) function gets random data from the RNGA module.

### 32.3.5 Seed RNGA

1. [RNGA\\_DRV\\_Seed\(\)](#) function inputs an entropy value that the RNGA can use to seed the pseudo random algorithm.

### 32.3.6 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

- RNG\_Type \*const `g_rngaBase` []  
*Table of base addresses for RNGA instances.*
- const IRQn\_Type `g_rngaIrqId` [RNG\_INSTANCE\_COUNT]  
*Table to save RNGA IRQ enumeration numbers defined in the CMSIS header file.*

### 32.3.7 Data Structure Documentation

#### 32.3.7.1 struct `rnga_user_config_t`

This structure initializes the RNGA by calling the 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.*

### 32.3.8 Function Documentation

#### 32.3.8.1 `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

#### 32.3.8.2 `void RNGA_DRV_Deinit ( uint32_t instance )`

This function shuts down the RNGA.

Parameters

|                      |             |
|----------------------|-------------|
| <i>instance,RNGA</i> | instance ID |
|----------------------|-------------|

#### 32.3.8.3 `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 |

### 32.3.8.4 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

### 32.3.8.5 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

### 32.3.8.6 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  |

**32.3.8.7 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 |
|----------------------|-------------|

**32.3.9 Variable Documentation**

**32.3.9.1 RNG\_Type\* const g\_rngaBase[]**

**32.3.9.2 const IRQn\_Type g\_rngaIrqId[RNG\_INSTANCE\_COUNT]**





## Chapter 33

# Real Time Clock (RTC)

### 33.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Real Time Clock (RTC) block of Kinetis devices.

NOTE: On platforms that do not have a 32KHz clock input to the RTC module, the RTC alarms may be inaccurate by a few seconds. Platforms that use the 1KHz LPO clock as input to the RTC will notice the date and time will drift away from wall clock as the LPO clock is not guaranteed to give exactly 1KHz (details can be found in the datasheet)

### Modules

- [RTC HAL driver](#)
- [RTC Peripheral Driver](#)

## RTC HAL driver

### 33.2 RTC HAL driver

#### 33.2.1 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.

#### 33.2.2 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 32 KHz clock from the system oscillator if the platform does not have an RTC oscillator. It also provides an initialization function ([RTC\\_HAL\\_Init\(\)](#)) to reset the RTC module.

#### 33.2.3 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. These are the structure details:

```
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 32 KHz, the [RTC\\_HAL\\_SetDatetime\(\)](#) and [RTC\\_HAL\\_GetDatetime\(\)](#) functions do not provide accurate results. On such boards, use the [RTC\\_HAL\\_SetDatetimeInsecs\(\)](#) and [RTC\\_HAL\\_GetDatetimeInsecs\(\)](#) functions and adjust the second 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 32 KHz. Use the peripheral driver instead of calling the HAL functions directly.

#### 33.2.4 IRTC 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 [irtc\\_datetime\\_t](#)  
*Structure is used to hold the time in a simple "date" format. [More...](#)*
- struct [irtc\\_daylight\\_time\\_t](#)  
*Structure is used to hold the daylight saving time. [More...](#)*
- struct [rtc\\_datetime\\_t](#)  
*Structure is used to hold the time in a simple "date" format. [More...](#)*

## Enumerations

- enum [irtc\\_clock\\_output\\_t](#) {  
  [kIRTCToOutputClk](#) = 0x0,  
  [kIRTCFine1hzClk](#) = 0x1,  
  [kIRTC32khzClk](#) = 0x2,  
  [kIRTCCoarse1hzClk](#) = 0x3 }  
*IRTC clock output selection.*
- enum [irtc\\_alarm\\_match\\_t](#) {  
  [kIRTCToSecMinHour](#) = 0x0,  
  [kIRTCToSecMinHourDay](#) = 0x1,  
  [kIRTCToSecMinHourDayMon](#) = 0x2,  
  [kIRTCToSecMinHourDayMonYear](#) = 0x3 }  
*IRTC alarm match selection.*
- enum [irtc\\_status\\_flag\\_t](#)  
*IRTC status flags.*
- enum [irtc\\_int\\_status\\_flag\\_t](#)  
*IRTC interrupt status flags.*
- enum [irtc\\_int\\_t](#)  
*IRTC interrupts.*
- enum [rtc\\_status\\_t](#) {  
  [kStatusRtcSuccess](#) = 0x00U,  
  [kStatusRtcFail](#) = 0x01U }  
*Error codes for RTC driver.*

## Initialization

- void [IRTC\\_HAL\\_Init](#) (RTC\_Type \*base, const [irtc\\_datetime\\_t](#) \*datetime, const [irtc\\_datetime\\_t](#) \*alarmDatetime, [irtc\\_alarm\\_match\\_t](#) alarmMode, const [irtc\\_daylight\\_time\\_t](#) \*daylightTime)  
*Resets the IRTC module.*
- void [IRTC\\_HAL\\_SetLockRegisterCmd](#) (RTC\_Type \*base, bool lock)  
*Lock or unlock IRTC registers for write access.*

## Date and Time Setting

## RTC HAL driver

### Note

All APIs of Date and Time Settings except `IRTC_HAL_SetDatetime` and `IRTC_HAL_GetDatetime` should first determine the state of the `INVAL` bit in the `STATUS`(bit 0) to determine the counters are stable before their value can be read or changed(can be done by calling `IRTC_HAL_GetStatus-Flag(base, kIRTCInvalidate)`). The assertion of `INVAL` bit ensures that no operation is done at the boundary of a second when counters change value.

- void `IRTC_HAL_SetYearMonth` (`RTC_Type *base`, `uint16_t year`, `uint16_t month`)  
*Sets the IRTC year and month.*
- void `IRTC_HAL_GetYearMonth` (`RTC_Type *base`, `uint16_t *year`, `uint16_t *month`)  
*Gets the IRTC current year and month.*
- static void `IRTC_HAL_SetDayWeek` (`RTC_Type *base`, `uint16_t day`, `uint16_t weekDay`)  
*Sets the IRTC day and day of week.*
- void `IRTC_HAL_GetDayWeek` (`RTC_Type *base`, `uint16_t *day`, `uint16_t *weekDay`)  
*Gets the IRTC current day and day of week.*
- static void `IRTC_HAL_SetHourMin` (`RTC_Type *base`, `uint16_t hour`, `uint16_t min`)  
*Sets the IRTC hour and min.*
- void `IRTC_HAL_GetHourMin` (`RTC_Type *base`, `uint16_t *hour`, `uint16_t *min`)  
*Gets the IRTC current hour and min.*
- static void `IRTC_HAL_SetSec` (`RTC_Type *base`, `uint16_t sec`)  
*Sets the IRTC second counter.*
- static void `IRTC_HAL_GetSec` (`RTC_Type *base`, `uint16_t *sec`)  
*Gets the IRTC current second counter.*
- void `IRTC_HAL_SetDatetime` (`RTC_Type *base`, `const irtc_datetime_t *datetime`)  
*Sets the IRTC date and time according to the given time structure.*
- void `IRTC_HAL_GetDatetime` (`RTC_Type *base`, `irtc_datetime_t *datetime`)  
*Gets the IRTC time and stores it in the given time structure.*

## Alarm Time Setting

- static void `IRTC_HAL_SetAlarmMatchMode` (`RTC_Type *base`, `irtc_alarm_match_t alarmType`)  
*Sets alarm match type.*
- void `IRTC_HAL_SetAlarmYearMonth` (`RTC_Type *base`, `uint16_t year`, `uint16_t month`)  
*Sets the IRTC year and month alarm.*
- void `IRTC_HAL_GetAlarmYearMonth` (`RTC_Type *base`, `uint16_t *year`, `uint16_t *month`)  
*Gets the IRTC current year and month alarm.*
- static void `IRTC_HAL_SetAlarmDay` (`RTC_Type *base`, `uint16_t day`)  
*Sets the IRTC day alarm.*
- static void `IRTC_HAL_GetAlarmDay` (`RTC_Type *base`, `uint16_t *day`)  
*Gets the IRTC current day alarm.*
- static void `IRTC_HAL_SetAlarmHourMin` (`RTC_Type *base`, `uint16_t hour`, `uint16_t min`)  
*Sets the IRTC hour and min alarm.*
- void `IRTC_HAL_GetAlarmHourMin` (`RTC_Type *base`, `uint16_t *hour`, `uint16_t *min`)  
*Gets the IRTC current hour and min alarm.*
- static void `IRTC_HAL_SetAlarmSec` (`RTC_Type *base`, `uint16_t sec`)  
*Sets the IRTC second counter alarm.*
- static void `IRTC_HAL_GetAlarmSec` (`RTC_Type *base`, `uint16_t *sec`)  
*Gets the IRTC current second counter alarm.*
- void `IRTC_HAL_SetAlarm` (`RTC_Type *base`, `const irtc_datetime_t *datetime`)

- *Sets the IRTC alarm time and enables the alarm interrupt.*
- void [IRTC\\_HAL\\_GetAlarm](#) (RTC\_Type \*base, [irtc\\_datetime\\_t](#) \*datetime)  
*Reads the value of the time alarm.*

## Daylight Saving Time Setting

- static void [IRTC\\_HAL\\_SetDaylightMonth](#) (RTC\_Type \*base, uint16\_t startMonth, uint16\_t endMonth)  
*Sets IRTC daylight saving start month and end month.*
- void [IRTC\\_HAL\\_GetDaylightMonth](#) (RTC\_Type \*base, uint16\_t \*startMonth, uint16\_t \*endMonth)  
*Gets the IRTC current daylight saving start and end month.*
- static void [IRTC\\_HAL\\_SetDaylightDay](#) (RTC\_Type \*base, uint16\_t startDay, uint16\_t endDay)  
*Sets the IRTC daylight saving start and end day.*
- void [IRTC\\_HAL\\_GetDaylightDay](#) (RTC\_Type \*base, uint16\_t \*startDay, uint16\_t \*endDay)  
*Gets the IRTC current daylight saving start and end day.*
- static void [IRTC\\_HAL\\_SetDaylightHour](#) (RTC\_Type \*base, uint16\_t startHour, uint16\_t endHour)  
*Sets the IRTC current daylight saving start and end hour.*
- void [IRTC\\_HAL\\_GetDaylightHour](#) (RTC\_Type \*base, uint16\_t \*startHour, uint16\_t \*endHour)  
*Gets the IRTC current daylight saving start and end hour.*
- void [IRTC\\_HAL\\_SetDaylightTime](#) (RTC\_Type \*base, const [irtc\\_daylight\\_time\\_t](#) \*datetime)  
*Sets the IRTC daylight saving date and time.*
- void [IRTC\\_HAL\\_GetDaylightTime](#) (RTC\_Type \*base, [irtc\\_daylight\\_time\\_t](#) \*datetime)  
*Gets the IRTC daylight saving time and stores it in the given time structure.*

## Status

- static bool [IRTC\\_HAL\\_GetStatusFlag](#) (RTC\_Type \*base, [irtc\\_status\\_flag\\_t](#) statusFlag)  
*Gets the IRTC status flag state.*

## Interrupts

- static void [IRTC\\_HAL\\_SetIntCmd](#) (RTC\_Type \*base, [irtc\\_int\\_t](#) interrupt, bool enable)  
*Enable or disable related IRTC interrupt.*
- static bool [IRTC\\_HAL\\_GetIntCmd](#) (RTC\_Type \*base, [irtc\\_int\\_t](#) interrupt)  
*Whether the IRTC interrupt is enabled or not.*
- static bool [IRTC\\_HAL\\_GetIntStatusFlag](#) (RTC\_Type \*base, [irtc\\_int\\_status\\_flag\\_t](#) statusFlag)  
*Gets the IRTC interrupt status flag state.*
- static void [IRTC\\_HAL\\_ClearIntStatusFlag](#) (RTC\_Type \*base, [irtc\\_int\\_status\\_flag\\_t](#) statusFlag)  
*Clear the IRTC interrupt status flag.*

## IRTC Time Compensation

- static void [IRTC\\_HAL\\_SetCompensation](#) (RTC\_Type \*base, bool enableFine, uint16\_t compValue)  
*Writes the value to the IRTC compensation register.*

## RTC HAL driver

- static uint16\_t [IRTC\\_HAL\\_GetCompensation](#) (RTC\_Type \*base)  
*Reads the time compensation register contents.*

## IRTC Control

- static void [IRTC\\_HAL\\_SetClockOutMode](#) (RTC\_Type \*base, [irtc\\_clock\\_output\\_t](#) clock)  
*Select which clock to output from SoC for use outside RTC.*
- static void [IRTC\\_HAL\\_SoftwareReset](#) (RTC\_Type \*base)  
*Performs a software reset on the IRTC module.*

## RTC HAL API Functions

- void [RTC\\_HAL\\_Enable](#) (RTC\_Type \*rtcBase)  
*Initializes the RTC module.*
- void [RTC\\_HAL\\_Disable](#) (RTC\_Type \*rtcBase)  
*Disables the RTC module.*
- void [RTC\\_HAL\\_Init](#) (RTC\_Type \*rtcBase)  
*This function will clear all interrupts.*
- 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](#) (RTC\_Type \*rtcBase, const [rtc\\_datetime\\_t](#) \*datetime)  
*Sets the RTC date and time according to the given time structure.*
- void [RTC\\_HAL\\_SetDatetimeInsecs](#) (RTC\_Type \*rtcBase, const uint32\_t seconds)  
*Sets the RTC date and time according to the given time provided in seconds.*
- void [RTC\\_HAL\\_GetDatetime](#) (RTC\_Type \*rtcBase, [rtc\\_datetime\\_t](#) \*datetime)  
*Gets the RTC time and stores it in the given time structure.*
- void [RTC\\_HAL\\_GetDatetimeInSecs](#) (RTC\_Type \*rtcBase, uint32\_t \*seconds)  
*Gets the RTC time and returns it in seconds.*
- void [RTC\\_HAL\\_GetAlarm](#) (RTC\_Type \*rtcBase, [rtc\\_datetime\\_t](#) \*date)  
*Reads the value of the time alarm.*
- bool [RTC\\_HAL\\_SetAlarm](#) (RTC\_Type \*rtcBase, 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](#) (RTC\_Type \*rtcBase)  
*Reads the value of the time seconds counter.*
- static void [RTC\\_HAL\\_SetSecsReg](#) (RTC\_Type \*rtcBase, const uint32\_t seconds)  
*Writes to the time seconds counter.*
- static void [RTC\\_HAL\\_SetAlarmReg](#) (RTC\_Type \*rtcBase, const uint32\_t seconds)  
*Sets the time alarm and clears the time alarm flag.*
- static uint32\_t [RTC\\_HAL\\_GetAlarmReg](#) (RTC\_Type \*rtcBase)  
*Gets the time alarm register contents.*

- static uint16\_t [RTC\\_HAL\\_GetPrescaler](#) (RTC\_Type \*rtcBase)  
*Reads the value of the time prescaler.*
- static void [RTC\\_HAL\\_SetPrescaler](#) (RTC\_Type \*rtcBase, const uint16\_t prescale)  
*Sets the time prescaler.*
- static uint32\_t [RTC\\_HAL\\_GetCompensationReg](#) (RTC\_Type \*rtcBase)  
*Reads the time compensation register contents.*
- static void [RTC\\_HAL\\_SetCompensationReg](#) (RTC\_Type \*rtcBase, const uint32\_t compValue)  
*Writes the value to the RTC TCR register.*
- static uint8\_t [RTC\\_HAL\\_GetCompensationIntervalCounter](#) (RTC\_Type \*rtcBase)  
*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](#) (RTC\_Type \*rtcBase)  
*Reads the current value used by the compensation logic for the present second interval.*
- static uint8\_t [RTC\\_HAL\\_GetCompensationIntervalRegister](#) (RTC\_Type \*rtcBase)  
*Reads the compensation interval register.*
- static void [RTC\\_HAL\\_SetCompensationIntervalRegister](#) (RTC\_Type \*rtcBase, const uint8\_t value)  
*Writes the compensation interval.*
- static uint8\_t [RTC\\_HAL\\_GetTimeCompensationRegister](#) (RTC\_Type \*rtcBase)  
*Reads the time compensation value which is the configured number of 32.768 kHz clock cycles in each second.*
- static void [RTC\\_HAL\\_SetTimeCompensationRegister](#) (RTC\_Type \*rtcBase, const uint8\_t compValue)  
*Writes to the field Time Compensation Register (TCR) of the RTC Time Compensation Register (RTC\_TCR).*
- static void [RTC\\_HAL\\_SetOsc2pfLoadCmd](#) (RTC\_Type \*rtcBase, bool enable)  
*Enables/disables the oscillator configuration for the 2pF load.*
- static bool [RTC\\_HAL\\_GetOsc2pfLoad](#) (RTC\_Type \*rtcBase)  
*Reads the oscillator 2pF load configure bit.*
- static void [RTC\\_HAL\\_SetOsc4pfLoadCmd](#) (RTC\_Type \*rtcBase, bool enable)  
*Enables/disables the oscillator configuration for the 4pF load.*
- static bool [RTC\\_HAL\\_GetOsc4pfLoad](#) (RTC\_Type \*rtcBase)  
*Reads the oscillator 4pF load configure bit.*
- static void [RTC\\_HAL\\_SetOsc8pfLoadCmd](#) (RTC\_Type \*rtcBase, bool enable)  
*Enables/disables the oscillator configuration for the 8pF load.*
- static bool [RTC\\_HAL\\_GetOsc8pfLoad](#) (RTC\_Type \*rtcBase)  
*Reads the oscillator 8pF load configure bit.*
- static void [RTC\\_HAL\\_SetOsc16pfLoadCmd](#) (RTC\_Type \*rtcBase, bool enable)  
*Enables/disables the oscillator configuration for the 16pF load.*
- static bool [RTC\\_HAL\\_GetOsc16pfLoad](#) (RTC\_Type \*rtcBase)  
*Reads the oscillator 16pF load configure bit.*
- static void [RTC\\_HAL\\_SetClockOutCmd](#) (RTC\_Type \*rtcBase, bool enable)  
*Enables/disables the 32 kHz clock output to other peripherals.*
- static bool [RTC\\_HAL\\_GetClockOutCmd](#) (RTC\_Type \*rtcBase)  
*Reads the RTC\_CR CLKO bit.*
- static void [RTC\\_HAL\\_SetOscillatorCmd](#) (RTC\_Type \*rtcBase, bool enable)  
*Enables/disables the oscillator.*
- static bool [RTC\\_HAL\\_IsOscillatorEnabled](#) (RTC\_Type \*rtcBase)  
*Reads the RTC\_CR OSCE bit.*
- static void [RTC\\_HAL\\_SoftwareReset](#) (RTC\_Type \*rtcBase)  
*Performs a software reset on the RTC module.*
- static void [RTC\\_HAL\\_SoftwareResetFlagClear](#) (RTC\_Type \*rtcBase)

## RTC HAL driver

- Clears the software reset flag.*

  - static bool [RTC\\_HAL\\_ReadSoftwareResetStatus](#) (RTC\_Type \*rtcBase)  
*Reads the RTC\_CR SWR bit.*
  - static bool [RTC\\_HAL\\_IsCounterEnabled](#) (RTC\_Type \*rtcBase)  
*Reads the time counter status (enabled/disabled).*
  - static void [RTC\\_HAL\\_EnableCounter](#) (RTC\_Type \*rtcBase, bool enable)  
*Changes the time counter status.*
  - static bool [RTC\\_HAL\\_HasAlarmOccured](#) (RTC\_Type \*rtcBase)  
*Checks whether the configured time alarm has occurred.*
  - static bool [RTC\\_HAL\\_IsTimeInvalid](#) (RTC\_Type \*rtcBase)  
*Checks whether the time has been marked as invalid.*
  - static bool [RTC\\_HAL\\_IsSecsIntEnabled](#) (RTC\_Type \*rtcBase)  
*Checks whether the Time Seconds Interrupt is enabled/disabled.*
  - static void [RTC\\_HAL\\_SetSecsIntCmd](#) (RTC\_Type \*rtcBase, bool enable)  
*Enables/disables the Time Seconds Interrupt.*
  - static bool [RTC\\_HAL\\_ReadAlarmInt](#) (RTC\_Type \*rtcBase)  
*Checks whether the Time Alarm Interrupt is enabled/disabled.*
  - static void [RTC\\_HAL\\_SetAlarmIntCmd](#) (RTC\_Type \*rtcBase, bool enable)  
*Enables/disables the Time Alarm Interrupt.*
  - static void [RTC\\_HAL\\_SetTimeOverflowIntCmd](#) (RTC\_Type \*rtcBase, bool enable)  
*Enables/disables the Time Overflow Interrupt.*
  - static void [RTC\\_HAL\\_SetTimeInvalidIntCmd](#) (RTC\_Type \*rtcBase, bool enable)  
*Enables/disables the Time Invalid Interrupt.*

## 33.2.5 Data Structure Documentation

### 33.2.5.1 struct irtc\_datetime\_t

#### Data Fields

- uint16\_t [year](#)  
*Range from 1984 to 2239.*
- uint16\_t [month](#)  
*Range from 1 to 12.*
- uint16\_t [day](#)  
*Range from 1 to 31 (depending on month).*
- uint16\_t [weekDay](#)  
*Range from 0(Sunday) to 6(Saturday)*
- uint16\_t [hour](#)  
*Range from 0 to 23.*
- uint16\_t [minute](#)  
*Range from 0 to 59.*
- uint16\_t [second](#)  
*Range from 0 to 59.*



### 33.2.5.1.0.54 Field Documentation

33.2.5.1.0.54.1 uint16\_t irtc\_datetime\_t::year

33.2.5.1.0.54.2 uint16\_t irtc\_datetime\_t::month

33.2.5.1.0.54.3 uint16\_t irtc\_datetime\_t::day

33.2.5.1.0.54.4 uint16\_t irtc\_datetime\_t::hour

33.2.5.1.0.54.5 uint16\_t irtc\_datetime\_t::minute

33.2.5.1.0.54.6 uint16\_t irtc\_datetime\_t::second

### 33.2.5.2 struct irtc\_daylight\_time\_t

#### Data Fields

- uint16\_t [startMonth](#)  
*Range from 1 to 12.*
- uint16\_t [endMonth](#)  
*Range from 1 to 12.*
- uint16\_t [startDay](#)  
*Range from 1 to 31 (depending on month).*
- uint16\_t [endDay](#)  
*Range from 1 to 31 (depending on month).*
- uint16\_t [startHour](#)  
*Range from 0 to 23.*
- uint16\_t [endHour](#)  
*Range from 0 to 23.*

### 33.2.5.2.0.55 Field Documentation

33.2.5.2.0.55.1 uint16\_t irtc\_daylight\_time\_t::startMonth

33.2.5.2.0.55.2 uint16\_t irtc\_daylight\_time\_t::endMonth

33.2.5.2.0.55.3 uint16\_t irtc\_daylight\_time\_t::startDay

33.2.5.2.0.55.4 uint16\_t irtc\_daylight\_time\_t::endDay

33.2.5.2.0.55.5 uint16\_t irtc\_daylight\_time\_t::startHour

33.2.5.2.0.55.6 uint16\_t irtc\_daylight\_time\_t::endHour

### 33.2.5.3 struct rtc\_datetime\_t

#### Data Fields

- uint16\_t [year](#)  
*Range from 1970 to 2099.*

## RTC HAL driver

- 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.

### 33.2.5.3.0.56 Field Documentation

33.2.5.3.0.56.1 uint16\_t rtc\_datetime\_t::year

33.2.5.3.0.56.2 uint16\_t rtc\_datetime\_t::month

33.2.5.3.0.56.3 uint16\_t rtc\_datetime\_t::day

33.2.5.3.0.56.4 uint16\_t rtc\_datetime\_t::hour

33.2.5.3.0.56.5 uint16\_t rtc\_datetime\_t::minute

33.2.5.3.0.56.6 uint8\_t rtc\_datetime\_t::second

## 33.2.6 Enumeration Type Documentation

### 33.2.6.1 enum irtc\_clock\_output\_t

Enumerator

- kIRTCTNoOutputClk* No output clock.
- kIRTCTFine1hzClk* Fine 1Hz clock.
- kIRTCT32khzClk* 32.768 kHz clock
- kIRTCTCoarse1hzClk* Coarse 1Hz clock.

### 33.2.6.2 enum irtc\_alarm\_match\_t

Enumerator

- kIRTCTSecMinHour* Generate alarm when sec/min/hour are matched.
- kIRTCTSecMinHourDay* Generate alarm when sec/min/hour/day are matched.
- kIRTCTSecMinHourDayMon* Generate alarm when sec/min/hour/day/month are matched.
- kIRTCTSecMinHourDayMonYear* Generate alarm when sec/min/hour/day/month/year are matched.

**33.2.6.3 enum irtc\_status\_flag\_t**

**33.2.6.4 enum irtc\_int\_status\_flag\_t**

**33.2.6.5 enum irtc\_int\_t**

**33.2.6.6 enum rtc\_status\_t**

Enumerator

*kStatusRtcSuccess* RTC success status.

*kStatusRtcFail* RTC error status.

**33.2.7 Function Documentation**

**33.2.7.1 void IRTC\_HAL\_Init ( RTC\_Type \* *base*, const irtc\_datetime\_t \* *datetime*, const irtc\_datetime\_t \* *alarmDatetime*, irtc\_alarm\_match\_t *alarmMode*, const irtc\_daylight\_time\_t \* *daylightTime* )**

This function initiates a soft-reset of the IRTC module to reset the IRTC registers, and configure IRTC according to user settings.

Parameters

|                      |                                                          |
|----------------------|----------------------------------------------------------|
| <i>base</i>          | The IRTC base address pointer.                           |
| <i>datetime</i>      | Date and time need to set, pass NULL to ignore.          |
| <i>alarmDatetime</i> | Alarm of date and time need to set, pass NULL to ignore. |
| <i>alarmMode</i>     | Alarm mode to set when will generate alarm.              |
| <i>daylightTime</i>  | Daylight saving time need to set, pass NULL to ignore.   |

**33.2.7.2 void IRTC\_HAL\_SetLockRegisterCmd ( RTC\_Type \* *base*, bool *lock* )**

Note

When the registers are unlocked, they remain in this unlocked state for a time of 2 seconds after which they get locked automatically. After power-on-reset, the registers come out as unlocked but they get locked automatically 15 seconds after power on.

## RTC HAL driver

### Parameters

|             |                                             |
|-------------|---------------------------------------------|
| <i>base</i> | The IRTC base address pointer.              |
| <i>lock</i> | Lock(true) or unlock(false) IRTC registers. |

### 33.2.7.3 void IRTC\_HAL\_SetYearMonth ( RTC\_Type \* *base*, uint16\_t *year*, uint16\_t *month* )

### Parameters

|              |                                |
|--------------|--------------------------------|
| <i>base</i>  | The IRTC base address pointer  |
| <i>year</i>  | Year number from 1984 to 2239. |
| <i>month</i> | Month number from 1 to 12.     |

### 33.2.7.4 void IRTC\_HAL\_GetYearMonth ( RTC\_Type \* *base*, uint16\_t \* *year*, uint16\_t \* *month* )

### Parameters

|              |                                        |
|--------------|----------------------------------------|
| <i>base</i>  | The IRTC base address pointer          |
| <i>year</i>  | Current year number from 1984 to 2239. |
| <i>month</i> | Current month number from 1 to 12.     |

### 33.2.7.5 static void IRTC\_HAL\_SetDayWeek ( RTC\_Type \* *base*, uint16\_t *day*, uint16\_t *weekDay* ) [inline], [static]

### Parameters

|                |                                                   |
|----------------|---------------------------------------------------|
| <i>base</i>    | The IRTC base address pointer                     |
| <i>day</i>     | Day number from 1 to 31.                          |
| <i>weekDay</i> | Day of week number from 0(Sunday) to 6(Saturday). |

### 33.2.7.6 void IRTC\_HAL\_GetDayWeek ( RTC\_Type \* *base*, uint16\_t \* *day*, uint16\_t \* *weekDay* )

Parameters

|                |                                                           |
|----------------|-----------------------------------------------------------|
| <i>base</i>    | The IRTC base address pointer                             |
| <i>day</i>     | Current day number from 1 to 31.                          |
| <i>weekDay</i> | Current day of week number from 0(Sunday) to 6(Saturday). |

**33.2.7.7** `static void IRTC_HAL_SetHourMin ( RTC_Type * base, uint16_t hour, uint16_t min ) [inline], [static]`

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | The IRTC base address pointer |
| <i>hour</i> | Hour number from 0 to 23.     |
| <i>min</i>  | Min number from 0 to 59.      |

**33.2.7.8** `void IRTC_HAL_GetHourMin ( RTC_Type * base, uint16_t * hour, uint16_t * min )`

Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>base</i> | The IRTC base address pointer     |
| <i>hour</i> | Current hour number from 0 to 23. |
| <i>min</i>  | Current min number from 0 to 59.  |

**33.2.7.9** `static void IRTC_HAL_SetSec ( RTC_Type * base, uint16_t sec ) [inline], [static]`

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | The IRTC base address pointer |
| <i>sec</i>  | Second number from 0 to 59.   |

**33.2.7.10** `static void IRTC_HAL_GetSec ( RTC_Type * base, uint16_t * sec ) [inline], [static]`

## RTC HAL driver

Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>base</i> | The IRTC base address pointer       |
| <i>sec</i>  | Current second number from 0 to 59. |

### 33.2.7.11 void IRTC\_HAL\_SetDatetime ( RTC\_Type \* *base*, const irtc\_datetime\_t \* *datetime* )

Parameters

|                 |                                                                    |
|-----------------|--------------------------------------------------------------------|
| <i>base</i>     | The IRTC base address pointer                                      |
| <i>datetime</i> | Pointer to a structure where the date and time details are stored. |

### 33.2.7.12 void IRTC\_HAL\_GetDatetime ( RTC\_Type \* *base*, irtc\_datetime\_t \* *datetime* )

Parameters

|                 |                                                                    |
|-----------------|--------------------------------------------------------------------|
| <i>base</i>     | The IRTC base address pointer                                      |
| <i>datetime</i> | Pointer to a structure where the date and time details are stored. |

### 33.2.7.13 static void IRTC\_HAL\_SetAlarmMatchMode ( RTC\_Type \* *base*, irtc\_alarm\_match\_t *alarmType* ) [inline], [static]

Parameters

|                  |                                                        |
|------------------|--------------------------------------------------------|
| <i>base</i>      | The IRTC base address pointer                          |
| <i>alarmType</i> | Alarm match selections that when an alarm will happen. |

### 33.2.7.14 void IRTC\_HAL\_SetAlarmYearMonth ( RTC\_Type \* *base*, uint16\_t *year*, uint16\_t *month* )

Parameters

---

|              |                                |
|--------------|--------------------------------|
| <i>base</i>  | The IRTC base address pointer  |
| <i>year</i>  | Year number from 1984 to 2239. |
| <i>month</i> | Month number from 1 to 12.     |

**33.2.7.15** `void IRTC_HAL_GetAlarmYearMonth ( RTC_Type * base, uint16_t * year, uint16_t * month )`

Parameters

|              |                                        |
|--------------|----------------------------------------|
| <i>base</i>  | The IRTC base address pointer          |
| <i>year</i>  | Current year number from 1984 to 2239. |
| <i>month</i> | Current month number from 1 to 12.     |

**33.2.7.16** `static void IRTC_HAL_SetAlarmDay ( RTC_Type * base, uint16_t day ) [inline], [static]`

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | The IRTC base address pointer |
| <i>day</i>  | Day number from 1 to 31.      |

**33.2.7.17** `static void IRTC_HAL_GetAlarmDay ( RTC_Type * base, uint16_t * day ) [inline], [static]`

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | The IRTC base address pointer         |
| <i>day</i>  | Current day number from 1984 to 2239. |

**33.2.7.18** `static void IRTC_HAL_SetAlarmHourMin ( RTC_Type * base, uint16_t hour, uint16_t min ) [inline], [static]`

## RTC HAL driver

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | The IRTC base address pointer |
| <i>hour</i> | Hour number from 0 to 23.     |
| <i>min</i>  | Min number from 0 to 59.      |

**33.2.7.19** `void IRTC_HAL_GetAlarmHourMin ( RTC_Type * base, uint16_t * hour, uint16_t * min )`

Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>base</i> | The IRTC base address pointer     |
| <i>hour</i> | Current hour number from 0 to 23. |
| <i>min</i>  | Current min number from 0 to 59.  |

**33.2.7.20** `static void IRTC_HAL_SetAlarmSec ( RTC_Type * base, uint16_t sec )`  
`[inline], [static]`

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | The IRTC base address pointer |
| <i>sec</i>  | Second number from 0 to 59.   |

**33.2.7.21** `static void IRTC_HAL_GetAlarmSec ( RTC_Type * base, uint16_t * sec )`  
`[inline], [static]`

Parameters

|             |                                     |
|-------------|-------------------------------------|
| <i>base</i> | The IRTC base address pointer       |
| <i>sec</i>  | Current second number from 0 to 59. |

**33.2.7.22** `void IRTC_HAL_SetAlarm ( RTC_Type * base, const irtc_datetime_t * datetime )`

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>base</i>     | The IRTC base address pointer.                                                |
| <i>datetime</i> | Pointer to structure where the alarm date and time details will be stored at. |

**33.2.7.23 void IRTC\_HAL\_GetAlarm ( RTC\_Type \* *base*, irtc\_datetime\_t \* *datetime* )**

Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>base</i>     | The IRTC base address pointer                                           |
| <i>datetime</i> | Pointer to a variable where the alarm date and time details are stored. |

**33.2.7.24 static void IRTC\_HAL\_SetDaylightMonth ( RTC\_Type \* *base*, uint16\_t *startMonth*, uint16\_t *endMonth* ) [inline],[static]**

Parameters

|                   |                                                  |
|-------------------|--------------------------------------------------|
| <i>base</i>       | The IRTC base address pointer                    |
| <i>startMonth</i> | Daylight saving start month number from 1 to 12. |
| <i>endMonth</i>   | Daylight saving end month number from 1 to 12.   |

**33.2.7.25 void IRTC\_HAL\_GetDaylightMonth ( RTC\_Type \* *base*, uint16\_t \* *startMonth*, uint16\_t \* *endMonth* )**

Parameters

|                   |                                                          |
|-------------------|----------------------------------------------------------|
| <i>base</i>       | The IRTC base address pointer                            |
| <i>startMonth</i> | Current daylight saving start month number from 1 to 12. |
| <i>endMonth</i>   | Current daylight saving end month number from 1 to 12.   |

**33.2.7.26 static void IRTC\_HAL\_SetDaylightDay ( RTC\_Type \* *base*, uint16\_t *startDay*, uint16\_t *endDay* ) [inline],[static]**

## RTC HAL driver

Parameters

|                 |                                                      |
|-----------------|------------------------------------------------------|
| <i>base</i>     | The IRTC base address pointer                        |
| <i>startDay</i> | Daylight saving day start value number from 1 to 31. |
| <i>endDay</i>   | Daylight saving day end value number from 1 to 31.   |

**33.2.7.27** void IRTC\_HAL\_GetDaylightDay ( RTC\_Type \* *base*, uint16\_t \* *startDay*, uint16\_t \* *endDay* )

Parameters

|                 |                                                              |
|-----------------|--------------------------------------------------------------|
| <i>base</i>     | The IRTC base address pointer                                |
| <i>startDay</i> | Current daylight saving day start value number from 1 to 31. |
| <i>endDay</i>   | Current daylight saving day end value number from 1 to 31.   |

**33.2.7.28** static void IRTC\_HAL\_SetDaylightHour ( RTC\_Type \* *base*, uint16\_t *startHour*, uint16\_t *endHour* ) [inline], [static]

Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>base</i>      | The IRTC base address pointer     |
| <i>startHour</i> | Daylight saving hour start value. |
| <i>endHour</i>   | Daylight saving hour end value.   |

**33.2.7.29** void IRTC\_HAL\_GetDaylightHour ( RTC\_Type \* *base*, uint16\_t \* *startHour*, uint16\_t \* *endHour* )

Parameters

|                  |                                           |
|------------------|-------------------------------------------|
| <i>base</i>      | The IRTC base address pointer             |
| <i>startHour</i> | Current daylight saving hour start value. |
| <i>endHour</i>   | Current daylight saving hour end value.   |

**33.2.7.30** void IRTC\_HAL\_SetDaylightTime ( RTC\_Type \* *base*, const irc\_daylight\_time\_t \* *datetime* )

Parameters

|                 |                                                                    |
|-----------------|--------------------------------------------------------------------|
| <i>base</i>     | The IRTC base address pointer                                      |
| <i>datetime</i> | Pointer to a structure where the date and time details are stored. |

**33.2.7.31** `void IRTC_HAL_GetDaylightTime ( RTC_Type * base, irtc_daylight_time_t * datetime )`

Parameters

|                 |                                                                    |
|-----------------|--------------------------------------------------------------------|
| <i>base</i>     | The IRTC base address pointer                                      |
| <i>datetime</i> | Pointer to a structure where the date and time details are stored. |

**33.2.7.32** `static bool IRTC_HAL_GetStatusFlag ( RTC_Type * base, irtc_status_flag_t statusFlag ) [inline], [static]`

*base* The IRTC peripheral base pointer.

Parameters

|                   |                                                      |
|-------------------|------------------------------------------------------|
| <i>statusFlag</i> | The status flag, defined in type irtc_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.

**33.2.7.33** `static void IRTC_HAL_SetIntCmd ( RTC_Type * base, irtc_int_t interrupt, bool enable ) [inline], [static]`

*base* The IRTC peripheral base pointer.

Parameters

|                  |                                                 |
|------------------|-------------------------------------------------|
| <i>interrupt</i> | The interrupt name, defined in type irtc_int_t. |
|------------------|-------------------------------------------------|

## RTC HAL driver

|               |                                                   |
|---------------|---------------------------------------------------|
| <i>enable</i> | Enable(true) or disable(false) related interrupt. |
|---------------|---------------------------------------------------|

### 33.2.7.34 `static bool IRTC_HAL_GetIntCmd ( RTC_Type * base, irtc_int_t interrupt ) [inline], [static]`

base The IRTC peripheral base pointer.

Parameters

|                  |                                                 |
|------------------|-------------------------------------------------|
| <i>interrupt</i> | The interrupt name, defined in type irtc_int_t. |
|------------------|-------------------------------------------------|

Returns

State of the interrupt: asserted (true) or not-asserted (false).

- true: related interrupt is being enabled.
- false: related interrupt is not enabled.

### 33.2.7.35 `static bool IRTC_HAL_GetIntStatusFlag ( RTC_Type * base, irtc_int_status_flag_t statusFlag ) [inline], [static]`

base The IRTC peripheral base pointer.

Parameters

|                   |                                                          |
|-------------------|----------------------------------------------------------|
| <i>statusFlag</i> | The status flag, defined in type irtc_int_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.

### 33.2.7.36 `static void IRTC_HAL_ClearIntStatusFlag ( RTC_Type * base, irtc_int_status_flag_t statusFlag ) [inline], [static]`

Tamper interrupt status flag is cleared when TAMPER\_SCR[TMPR\_STS] is cleared.

base The IRTC peripheral base pointer.

Parameters

|                   |                                                                        |
|-------------------|------------------------------------------------------------------------|
| <i>statusFlag</i> | The status flag, defined in type <code>irtc_int_status_flag_t</code> . |
|-------------------|------------------------------------------------------------------------|

**33.2.7.37** `static void IRTC_HAL_SetCompensation ( RTC_Type * base, bool enableFine, uint16_t compValue ) [inline], [static]`

Parameters

|                   |                                                   |
|-------------------|---------------------------------------------------|
| <i>base</i>       | The IRTC base address pointer                     |
| <i>enableFine</i> | Enable(true) or disable(false) fine compensation. |
| <i>compValue</i>  | value to be written to the compensation register. |

**33.2.7.38** `static uint16_t IRTC_HAL_GetCompensation ( RTC_Type * base ) [inline], [static]`

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | The IRTC base address pointer |
|-------------|-------------------------------|

Returns

time compensation register contents.

**33.2.7.39** `static void IRTC_HAL_SetClockOutMode ( RTC_Type * base, irtc_clock_output_t clock ) [inline], [static]`

Parameters

|              |                                                                        |
|--------------|------------------------------------------------------------------------|
| <i>base</i>  | The IRTC base address pointer.                                         |
| <i>clock</i> | OUtput clock selection from list of <code>irtc_clock_output_t</code> . |

**33.2.7.40** `static void IRTC_HAL_SoftwareReset ( RTC_Type * base ) [inline], [static]`

This clears the contents of alarm, interrupt(status and enable except tamper interrupt enable bit) registers, STATUS[`CMP_DONE`], and STATUS[`BUS_ERR`] and has no effect on DST, calendaring, standby time and tamper detect registers.

## RTC HAL driver

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | The IRTC base address pointer |
|-------------|-------------------------------|

### 33.2.7.41 void RTC\_HAL\_Enable ( RTC\_Type \* *rtcBase* )

This function enables the RTC oscillator.

Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

### 33.2.7.42 void RTC\_HAL\_Disable ( RTC\_Type \* *rtcBase* )

This function disables the RTC counter and oscillator.

Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

### 33.2.7.43 void RTC\_HAL\_Init ( RTC\_Type \* *rtcBase* )

This function initiates a soft-reset of the RTC module if the time invalid flag is set.

Parameters

|                |                               |
|----------------|-------------------------------|
| <i>rtcBase</i> | The RTC base address pointer. |
|----------------|-------------------------------|

### 33.2.7.44 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 |

### 33.2.7.45 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

**33.2.7.46 void RTC\_HAL\_ConvertDatetimeToSecs ( const rtc\_datetime\_t \* *datetime*, uint32\_t \* *seconds* )**

Parameters

|                 |                                                                           |
|-----------------|---------------------------------------------------------------------------|
| <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                              |

**33.2.7.47 void RTC\_HAL\_SetDatetime ( RTC\_Type \* *rtcBase*, 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>rtcBase</i>  | The RTC base address pointer                                                 |
| <i>datetime</i> | [in] Pointer to structure where the date and time details to set are stored. |

**33.2.7.48 void RTC\_HAL\_SetDatetimelnsecs ( RTC\_Type \* *rtcBase*, const uint32\_t *seconds* )**

The RTC counter is started after setting the time.

Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
| <i>seconds</i> | [in] Time in seconds         |

## RTC HAL driver

**33.2.7.49 void RTC\_HAL\_GetDatetime ( RTC\_Type \* *rtcBase*, 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>rtcBase</i>  | The RTC base address pointer                                             |
| <i>datetime</i> | [out] pointer to a structure where the date and time details are stored. |

### 33.2.7.50 void RTC\_HAL\_GetDatetimelnSecs ( RTC\_Type \* *rtcBase*, uint32\_t \* *seconds* )

Parameters

|                |                                                                   |
|----------------|-------------------------------------------------------------------|
| <i>rtcBase</i> | The RTC base address pointer                                      |
| <i>seconds</i> | [out] pointer to variable where the RTC time is stored in seconds |

### 33.2.7.51 void RTC\_HAL\_GetAlarm ( RTC\_Type \* *rtcBase*, rtc\_datetime\_t \* *date* )

Parameters

|                |                                                                               |
|----------------|-------------------------------------------------------------------------------|
| <i>rtcBase</i> | The RTC base address pointer                                                  |
| <i>date</i>    | [out] pointer to a variable where the alarm date and time details are stored. |

### 33.2.7.52 bool RTC\_HAL\_SetAlarm ( RTC\_Type \* *rtcBase*, 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>rtcBase</i> | The RTC base address pointer.                                                      |
| <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.

### 33.2.7.53 static uint32\_t RTC\_HAL\_GetSecsReg ( RTC\_Type \* *rtcBase* ) [inline], [static]

The time counter reads as zero if either the SR[TOF] or the SR[TIF] is set.

## RTC HAL driver

### Parameters

|                |                               |
|----------------|-------------------------------|
| <i>rtcBase</i> | The RTC base address pointer. |
|----------------|-------------------------------|

### Returns

contents of the seconds register.

#### 33.2.7.54 **static void RTC\_HAL\_SetSecsReg ( RTC\_Type \* *rtcBase*, 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>rtcBase</i> | The RTC base address pointer. |
| <i>seconds</i> | [in] seconds value.           |

#### 33.2.7.55 **static void RTC\_HAL\_SetAlarmReg ( RTC\_Type \* *rtcBase*, 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>rtcBase</i> | The RTC base address pointer. |
| <i>seconds</i> | [in] alarm value in seconds.  |

#### 33.2.7.56 **static uint32\_t RTC\_HAL\_GetAlarmReg ( RTC\_Type \* *rtcBase* ) [inline], [static]**

Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

Returns

contents of the alarm register.

**33.2.7.57 static uint16\_t RTC\_HAL\_GetPrescaler ( RTC\_Type \* *rtcBase* ) [inline], [static]**

The time counter reads as zero when either the SR[TOF] or the SR[TIF] is set.

Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

Returns

contents of the time prescaler register.

**33.2.7.58 static void RTC\_HAL\_SetPrescaler ( RTC\_Type \* *rtcBase*, 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>rtcBase</i>  | The RTC base address pointer |
| <i>prescale</i> | Prescaler value              |

**33.2.7.59 static uint32\_t RTC\_HAL\_GetCompensationReg ( RTC\_Type \* *rtcBase* ) [inline], [static]**

## RTC HAL driver

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

### Returns

time compensation register contents.

**33.2.7.60** `static void RTC_HAL_SetCompensationReg ( RTC_Type * rtcBase, const uint32_t compValue ) [inline], [static]`

### Parameters

|                  |                                                   |
|------------------|---------------------------------------------------|
| <i>rtcBase</i>   | The RTC base address pointer                      |
| <i>compValue</i> | value to be written to the compensation register. |

**33.2.7.61** `static uint8_t RTC_HAL_GetCompensationIntervalCounter ( RTC_Type * rtcBase ) [inline], [static]`

### Parameters

|                |                               |
|----------------|-------------------------------|
| <i>rtcBase</i> | The RTC base address pointer. |
|----------------|-------------------------------|

### Returns

compensation interval value.

**33.2.7.62** `static uint8_t RTC_HAL_GetTimeCompensationValue ( RTC_Type * rtcBase ) [inline], [static]`

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

### Returns

time compensation value

**33.2.7.63** `static uint8_t RTC_HAL_GetCompensationIntervalRegister ( RTC_Type * rtcBase ) [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>rtcBase</i> | The RTC base address pointer. |
|----------------|-------------------------------|

Returns

compensation interval in seconds.

**33.2.7.64** `static void RTC_HAL_SetCompensationIntervalRegister ( RTC_Type * rtcBase, 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>rtcBase</i> | The RTC base address pointer.    |
| <i>value</i>   | the compensation interval value. |

**33.2.7.65** `static uint8_t RTC_HAL_GetTimeCompensationRegister ( RTC_Type * rtcBase ) [inline], [static]`

Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

Returns

time compensation value.

## RTC HAL driver

**33.2.7.66** `static void RTC_HAL_SetTimeCompensationRegister ( RTC_Type * rtcBase,  
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.

## Parameters

|                  |                                 |
|------------------|---------------------------------|
| <i>rtcBase</i>   | The RTC base address pointer    |
| <i>compValue</i> | value of the time compensation. |

**33.2.7.67** `static void RTC_HAL_SetOsc2pfLoadCmd ( RTC_Type * rtcBase, bool enable ) [inline], [static]`

## Parameters

|                |                                                                 |
|----------------|-----------------------------------------------------------------|
| <i>rtcBase</i> | The RTC base address pointer                                    |
| <i>enable</i>  | can be true or false -true: enables load -false: disables load. |

**33.2.7.68** `static bool RTC_HAL_GetOsc2pfLoad ( RTC_Type * rtcBase ) [inline], [static]`

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

## Returns

true: 2pF additional load enabled. false: 2pF additional load disabled.

**33.2.7.69** `static void RTC_HAL_SetOsc4pfLoadCmd ( RTC_Type * rtcBase, bool enable ) [inline], [static]`

## Parameters

|                |                                                                 |
|----------------|-----------------------------------------------------------------|
| <i>rtcBase</i> | The RTC base address pointer                                    |
| <i>enable</i>  | can be true or false -true: enables load. -false: disables load |

**33.2.7.70** `static bool RTC_HAL_GetOsc4pfLoad ( RTC_Type * rtcBase ) [inline], [static]`

## RTC HAL driver

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

### Returns

true: 4pF additional load enabled. false: 4pF additional load disabled.

**33.2.7.71** `static void RTC_HAL_SetOsc8pfLoadCmd ( RTC_Type * rtcBase, bool enable ) [inline], [static]`

### Parameters

|                |                                                                  |
|----------------|------------------------------------------------------------------|
| <i>rtcBase</i> | The RTC base address pointer                                     |
| <i>enable</i>  | can be true or false -true: enables load. -false: disables load. |

**33.2.7.72** `static bool RTC_HAL_GetOsc8pfLoad ( RTC_Type * rtcBase ) [inline], [static]`

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

### Returns

true: 8pF additional load enabled. false: 8pF additional load disabled.

**33.2.7.73** `static void RTC_HAL_SetOsc16pfLoadCmd ( RTC_Type * rtcBase, bool enable ) [inline], [static]`

### Parameters

|                |                                                                  |
|----------------|------------------------------------------------------------------|
| <i>rtcBase</i> | The RTC base address pointer                                     |
| <i>enable</i>  | can be true or false -true: enables load. -false: disables load. |

**33.2.7.74** `static bool RTC_HAL_GetOsc16pfLoad ( RTC_Type * rtcBase ) [inline], [static]`



Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

Returns

true: 16pF additional load enabled. false: 16pF additional load disabled.

**33.2.7.75 static void RTC\_HAL\_SetClockOutCmd ( RTC\_Type \* *rtcBase*, bool *enable* ) [inline], [static]**

Parameters

|                |                                                                            |
|----------------|----------------------------------------------------------------------------|
| <i>rtcBase</i> | The RTC base address pointer                                               |
| <i>enable</i>  | can be true or false -true: enables clock out. -false: disables clock out. |

**33.2.7.76 static bool RTC\_HAL\_GetClockOutCmd ( RTC\_Type \* *rtcBase* ) [inline], [static]**

Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

Returns

true: 32 kHz clock is not output to other peripherals. false: 32 kHz clock is output to other peripherals.

**33.2.7.77 static void RTC\_HAL\_SetOscillatorCmd ( RTC\_Type \* *rtcBase*, 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.

Parameters

|                |                                                                              |
|----------------|------------------------------------------------------------------------------|
| <i>rtcBase</i> | The RTC base address pointer                                                 |
| <i>enable</i>  | can be true or false -true: enables oscillator. -false: disables oscillator. |

**33.2.7.78 static bool RTC\_HAL\_IsOscillatorEnabled ( RTC\_Type \* *rtcBase* ) [inline], [static]**

## RTC HAL driver

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

### Returns

true: 32.768 kHz oscillator is enabled false: 32.768 kHz oscillator is disabled.

### 33.2.7.79 static void RTC\_HAL\_SoftwareReset ( RTC\_Type \* *rtcBase* ) [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.

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

### 33.2.7.80 static void RTC\_HAL\_SoftwareResetFlagClear ( RTC\_Type \* *rtcBase* ) [inline], [static]

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

### 33.2.7.81 static bool RTC\_HAL\_ReadSoftwareResetStatus ( RTC\_Type \* *rtcBase* ) [inline], [static]

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

### Returns

true: SWR is set. false: SWR is cleared.

### 33.2.7.82 static bool RTC\_HAL\_IsCounterEnabled ( RTC\_Type \* *rtcBase* ) [inline], [static]

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

## 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.

**33.2.7.83 static void RTC\_HAL\_EnableCounter ( RTC\_Type \* *rtcBase*, bool *enable* ) [inline], [static]**

## Parameters

|                |                                                                                         |
|----------------|-----------------------------------------------------------------------------------------|
| <i>rtcBase</i> | The RTC base address pointer                                                            |
| <i>enable</i>  | can be true or false -true: enables the time counter -false: disables the time counter. |

**33.2.7.84 static bool RTC\_HAL\_HasAlarmOccured ( RTC\_Type \* *rtcBase* ) [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>rtcBase</i> | The RTC base address pointer. |
|----------------|-------------------------------|

## Returns

-true: time alarm has occurred. -false: no time alarm occurred.

**33.2.7.85 static bool RTC\_HAL\_IsTimeInvalid ( RTC\_Type \* *rtcBase* ) [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 flag is cleared by writing the TSR register when the time counter is disabled.

## RTC HAL driver

### Parameters

|                |                               |
|----------------|-------------------------------|
| <i>rtcBase</i> | The RTC base address pointer. |
|----------------|-------------------------------|

### Returns

-true: time is INVALID and time counter is zero. -false: time is valid.

### 33.2.7.86 `static bool RTC_HAL_IsSecsIntEnabled ( RTC_Type * rtcBase ) [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>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

### Returns

-true: Seconds interrupt is enabled. -false: Seconds interrupt is disabled.

### 33.2.7.87 `static void RTC_HAL_SetSecsIntCmd ( RTC_Type * rtcBase, 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).

### Parameters

|                |                                                                                                  |
|----------------|--------------------------------------------------------------------------------------------------|
| <i>rtcBase</i> | The RTC base address pointer                                                                     |
| <i>enable</i>  | can be true or false -true: Seconds interrupt is enabled. -false: Seconds interrupt is disabled. |

### 33.2.7.88 `static bool RTC_HAL_ReadAlarmInt ( RTC_Type * rtcBase ) [inline], [static]`

Reads the field Time Alarm Interrupt Enable (TAIE) value of the RTC Interrupt Enable Register (RTC\_IER).

Parameters

|                |                              |
|----------------|------------------------------|
| <i>rtcBase</i> | The RTC base address pointer |
|----------------|------------------------------|

Returns

true: Time alarm flag does generate an interrupt. false: Time alarm flag does not generate an interrupt.

**33.2.7.89 static void RTC\_HAL\_SetAlarmIntCmd ( RTC\_Type \* *rtcBase*, bool *enable* ) [inline], [static]**

Writes to the field Time Alarm Interrupt Enable (TAIE) of the RTC Interrupt Enable Register (RTC\_IER).

Parameters

|                |                                                                                                                                 |
|----------------|---------------------------------------------------------------------------------------------------------------------------------|
| <i>rtcBase</i> | The RTC base address pointer                                                                                                    |
| <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. |

**33.2.7.90 static void RTC\_HAL\_SetTimeOverflowIntCmd ( RTC\_Type \* *rtcBase*, bool *enable* ) [inline], [static]**

Writes to the field Time Overflow Interrupt Enable (TOIE) of the RTC Interrupt Enable Register (RTC\_IER).

Parameters

|                |                                                                                                                                       |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>rtcBase</i> | The RTC base address pointer                                                                                                          |
| <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. |

**33.2.7.91 static void RTC\_HAL\_SetTimeInvalidIntCmd ( RTC\_Type \* *rtcBase*, 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>rtcBase</i> | The RTC base address pointer                                                                                                        |
| <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. |

## 33.3 RTC Peripheral Driver

### 33.3.1 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.

### 33.3.2 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);
```

### 33.3.3 IRTCS Setting and reading the IRTC 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);
```

### 33.3.4 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);
```

### 33.3.5 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();
}

```

### 33.3.6 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.

### 33.3.7 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\\_IRQHandler\(\)](#) 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

- RTC\_Type \*const [g\\_rtcBase](#) [RTC\_INSTANCE\_COUNT]  
*Table of base addresses for RTC instances.*
- const IRQn\_Type [g\\_rtcIrqId](#) [RTC\_INSTANCE\_COUNT]  
*Table to save RTC Alarm IRQ numbers for RTC instances.*
- const IRQn\_Type [g\\_rtcSecondsIrqId](#) [RTC\_INSTANCE\_COUNT]  
*Table to save RTC Seconds IRQ numbers for RTC instances.*

## Initialization and De-initialization

- [rtc\\_status\\_t 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)  
*Implements the RTC alarm handler named in the startup code.*
- void [RTC\\_Seconds\\_IRQHandler](#) (void)  
*Implements the RTC seconds handler named in the startup code.*

## 33.3.8 Data Structure Documentation

### 33.3.8.1 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.*

#### 33.3.8.1.0.57 Field Documentation

33.3.8.1.0.57.1 [rtc\\_datetime\\_t rtc\\_repeat\\_alarm\\_state\\_t::alarmTime](#)

33.3.8.1.0.57.2 [rtc\\_datetime\\_t rtc\\_repeat\\_alarm\\_state\\_t::alarmRepTime](#)

## 33.3.9 Function Documentation

### 33.3.9.1 [rtc\\_status\\_t RTC\\_DRV\\_Init \( uint32\\_t instance \)](#)

Enables the RTC clock and interrupts if requested by the user.

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The RTC peripheral instance number. |
|-----------------|-------------------------------------|

Returns

kStatusRtcSuccess means success, otherwise means failed.

### 33.3.9.2 void [RTC\\_DRV\\_Deinit \( uint32\\_t instance \)](#)

## RTC Peripheral Driver

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The RTC peripheral instance number. |
|-----------------|-------------------------------------|

### 33.3.9.3 bool RTC\_DRV\_IsCounterEnabled ( uint32\_t instance )

The function checks the TCE bit in the RTC control register.

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The RTC peripheral instance number. |
|-----------------|-------------------------------------|

Returns

true: The RTC counter is enabled  
false: The RTC counter is disabled

### 33.3.9.4 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.

### 33.3.9.5 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. |
|-----------------|------------------------------------------------------------------------|

### 33.3.9.6 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<br>true: indicates seconds interrupt should be enabled<br>false: indicates seconds interrupt should be disabled |

### 33.3.9.7 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 of false<br>true: indicates alarm interrupt should be enabled<br>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.

### 33.3.9.8 void RTC\_DRV\_GetAlarm ( uint32\_t *instance*, rtc\_datetime\_t \* *date* )

Parameters

## RTC Peripheral Driver

|                 |                                                                              |
|-----------------|------------------------------------------------------------------------------|
| <i>instance</i> | The RTC peripheral instance number.                                          |
| <i>date</i>     | [out] pointer to structure where the alarm date and time details are stored. |

### 33.3.9.9 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.

Parameters

|                               |                                                      |
|-------------------------------|------------------------------------------------------|
| <i>instance</i>               | The RTC peripheral instance number.                  |
| <i>repeatAlarm-<br/>State</i> | Pointer to structure where the alarm state is stored |

### 33.3.9.10 bool RTC\_DRV\_SetAlarmRepeat ( uint32\_t *instance*, rtc\_datetime\_t \* *alarmTime*, rtc\_datetime\_t \* *alarmReplInterval* )

Parameters

|                               |                                                        |
|-------------------------------|--------------------------------------------------------|
| <i>instance</i>               | The RTC peripheral instance number.                    |
| <i>alarmTime</i>              | Pointer to structure where the alarm time is provided. |
| <i>alarmRep-<br/>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.

### 33.3.9.11 void RTC\_DRV\_DeinitRepeatAlarm ( uint32\_t *instance* )

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The RTC peripheral instance number. |
|-----------------|-------------------------------------|

### 33.3.9.12 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<br>true: indicates alarm interrupt should be enabled<br>false: indicates alarm interrupt should be disabled |

### 33.3.9.13 `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

### 33.3.9.14 `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

### 33.3.9.15 `void RTC_DRV_SetTimeCompensation ( uint32_t instance, uint32_t compensationInterval, uint32_t compensationTime )`

#### Parameters

---

## RTC Peripheral Driver

|                              |                                                                                                               |
|------------------------------|---------------------------------------------------------------------------------------------------------------|
| <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)     |

### 33.3.9.16 void RTC\_DRV\_GetTimeCompensation ( uint32\_t *instance*, uint32\_t \* *compensationInterval*, uint32\_t \* *compensationTime* )

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)       |

### 33.3.9.17 void RTC\_IRQHandler ( void )

Handles the RTC alarm interrupt and invokes any callback that is interested in the RTC alarm.

### 33.3.9.18 void RTC\_Seconds\_IRQHandler ( void )

Handles the RTC seconds interrupt and invokes any callback that is interested in the RTC second tick.

### 33.3.9.19 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. |
|-----------------|-------------------------------------|

### 33.3.9.20 void RTC\_DRV\_SecsIntAction ( uint32\_t *instance* )

Disables the time seconds interrupt (TSIE) bit.



Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The RTC peripheral instance number. |
|-----------------|-------------------------------------|

### 33.3.10 Variable Documentation

**33.3.10.1** `RTC_Type* const g_rtcBase[RTC_INSTANCE_COUNT]`

**33.3.10.2** `const IRQn_Type g_rtcIrqId[RTC_INSTANCE_COUNT]`

**33.3.10.3** `const IRQn_Type g_rtcSecondsIrqId[RTC_INSTANCE_COUNT]`



## Chapter 34

# Synchronous Audio Interface (SAI)

### 34.1 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](#)

## SAI HAL driver

### 34.2 SAI HAL driver

#### 34.2.1 Overview

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

#### Files

- file [fsl\\_sai\\_hal.h](#)

#### Data Structures

- struct [sai\\_clock\\_setting\\_t](#)  
*SAI clock configuration structure. [More...](#)*

#### Enumerations

- enum [sai\\_protocol\\_t](#) {  
[kSaiBusI2SLeft](#) = 0x0u,  
[kSaiBusI2SRight](#) = 0x1u,  
[kSaiBusI2SType](#) = 0x2u,  
[kSaiBusPCMA](#) = 0x3u,  
[kSaiBusPCMB](#) = 0x4u,  
[kSaiBusAC97](#) = 0x5u }  
*Define the bus type of sai.*
- enum [sai\\_master\\_slave\\_t](#) {  
[kSaiMaster](#) = 0x0u,  
[kSaiSlave](#) = 0x1u }  
*Master or slave mode.*
- enum [sai\\_mono\\_stereo\\_t](#) {  
[kSaiMono](#) = 0x0u,  
[kSaiStereo](#) = 0x1u }
- enum [sai\\_sync\\_mode\\_t](#) {  
[kSaiModeAsync](#) = 0x0u,  
[kSaiModeSync](#) = 0x1u,  
[kSaiModeSyncWithOtherTx](#) = 0x2u,  
[kSaiModeSyncWithOtherRx](#) = 0x3u }  
*Synchronous or asynchronous mode.*
- enum [sai\\_mclk\\_source\\_t](#) {  
[kSaiMclkSourceSysclk](#) = 0x0u,  
[kSaiMclkSourceSelect1](#) = 0x1u,  
[kSaiMclkSourceSelect2](#) = 0x2u,  
[kSaiMclkSourceSelect3](#) = 0x3u }  
*Master clock source.*

- enum `sai_bclk_source_t` {  
`kSaiBclkSourceBusclk` = 0x0u,  
`kSaiBclkSourceMclkDiv` = 0x1u,  
`kSaiBclkSourceOtherSai0` = 0x2u,  
`kSaiBclkSourceOtherSai1` = 0x3u }  
*Bit clock source.*
- enum `sai_interrupt_request_t` {  
`kSaiIntrequestWordStart` = 0x1000u,  
`kSaiIntrequestSyncError` = 0x800u,  
`kSaiIntrequestFIFOWarning` = 0x200u,  
`kSaiIntrequestFIFOError` = 0x400u,  
`kSaiIntrequestFIFORequest` = 0x100u }  
*The SAI state flag.*
- enum `sai_dma_request_t` {  
`kSaiDmaReqFIFOWarning` = 0x2u,  
`kSaiDmaReqFIFORequest` = 0x1u }  
*The DMA request sources.*
- enum `sai_state_flag_t` {  
`kSaiStateFlagWordStart` = 0x100000u,  
`kSaiStateFlagSyncError` = 0x80000u,  
`kSaiStateFlagFIFOError` = 0x40000u,  
`kSaiStateFlagFIFORequest` = 0x10000u,  
`kSaiStateFlagFIFOWarning` = 0x20000u,  
`kSaiStateFlagSoftReset` = 0x1000000u,  
`kSaiStateFlagAll` = 0x11F0000u }  
*The SAI state flag.*
- enum `sai_reset_type_t` {  
`kSaiResetTypeSoftware` = 0x1000000u,  
`kSaiResetTypeFIFO` = 0x2000000u,  
`kSaiResetAll` = 0x3000000u }  
*The reset type.*
- enum `sai_run_mode_t` {  
`kSaiRunModeDebug` = 0x0,  
`kSaiRunModeStop` = 0x1 }

## Module control

- void `SAI_HAL_TxInit` (I2S\_Type \*base)  
*Initializes the SAI Tx.*
- void `SAI_HAL_RxInit` (I2S\_Type \*base)  
*Initializes the SAI Rx.*
- void `SAI_HAL_TxSetProtocol` (I2S\_Type \*base, `sai_protocol_t` protocol)  
*Sets Tx protocol relevant settings.*
- void `SAI_HAL_RxSetProtocol` (I2S\_Type \*base, `sai_protocol_t` protocol)  
*Sets Rx protocol relevant settings.*
- void `SAI_HAL_TxSetMasterSlave` (I2S\_Type \*base, `sai_master_slave_t` master\_slave\_mode)

## SAI HAL driver

- Sets master or slave mode.*

  - void [SAI\\_HAL\\_RxSetMasterSlave](#) (I2S\_Type \*base, [sai\\_master\\_slave\\_t](#) master\_slave\_mode)
 

*Sets master or slave mode.*

## Overall Clock configuration

- void [SAI\\_HAL\\_TxClockSetup](#) (I2S\_Type \*base, [sai\\_clock\\_setting\\_t](#) \*clk\_config)
 

*Setup clock for SAI Tx.*
- void [SAI\\_HAL\\_RxClockSetup](#) (I2S\_Type \*base, [sai\\_clock\\_setting\\_t](#) \*clk\_config)
 

*Setup clock for SAI Rx.*

## Master clock configuration

- static void [SAI\\_HAL\\_SetMclkSrc](#) (I2S\_Type \*base, [sai\\_mclk\\_source\\_t](#) source)
 

*Sets the master clock source.*
- static uint32\_t [SAI\\_HAL\\_GetMclkSrc](#) (I2S\_Type \*base)
 

*Gets the master clock source.*
- static void [SAI\\_HAL\\_SetMclkDividerCmd](#) (I2S\_Type \*base, bool enable)
 

*Enable or disable MCLK internal.*

## Bit clock configuration

- static void [SAI\\_HAL\\_TxSetBclkSrc](#) (I2S\_Type \*base, [sai\\_bclk\\_source\\_t](#) source)
 

*Sets the bit clock source of Tx.*
- static void [SAI\\_HAL\\_RxSetBclkSrc](#) (I2S\_Type \*base, [sai\\_bclk\\_source\\_t](#) source)
 

*Sets bit clock source of the Rx.*
- static uint32\_t [SAI\\_HAL\\_TxGetBclkSrc](#) (I2S\_Type \*base)
 

*Gets the bit clock source of Tx.*
- static uint32\_t [SAI\\_HAL\\_RxGetBclkSrc](#) (I2S\_Type \*base)
 

*Gets bit clock source of the Rx.*
- static void [SAI\\_HAL\\_TxSetBclkDiv](#) (I2S\_Type \*base, uint32\_t divider)
 

*Sets the Tx bit clock divider value.*
- static void [SAI\\_HAL\\_RxSetBclkDiv](#) (I2S\_Type \*base, uint32\_t divider)
 

*Sets the Rx bit clock divider value.*
- static void [SAI\\_HAL\\_TxSetBclkInputCmd](#) (I2S\_Type \*base, bool enable)
 

*Enables or disables the Tx bit clock input bit.*
- static void [SAI\\_HAL\\_RxSetBclkInputCmd](#) (I2S\_Type \*base, bool enable)
 

*Enables or disables the Rx bit clock input bit.*
- static void [SAI\\_HAL\\_TxSetSwapBclkCmd](#) (I2S\_Type \*base, bool enable)
 

*Sets the Tx bit clock swap.*
- static void [SAI\\_HAL\\_RxSetSwapBclkCmd](#) (I2S\_Type \*base, bool enable)
 

*Sets the Rx bit clock swap.*

## Mono or stereo configuration

- void [SAI\\_HAL\\_TxSetMonoStereo](#) (I2S\_Type \*base, [sai\\_mono\\_stereo\\_t](#) mono\_stereo)

- *Set Tx audio channel number.*
- void [SAI\\_HAL\\_RxSetMonoStereo](#) (I2S\_Type \*base, [sai\\_mono\\_stereo\\_t](#) mono\_stereo)  
*Set Rx audio channel number.*

## Word configurations

- void [SAI\\_HAL\\_TxSetWordWidth](#) (I2S\_Type \*base, [sai\\_protocol\\_t](#) protocol, uint32\_t bits)  
*Set Tx word width.*
- void [SAI\\_HAL\\_RxSetWordWidth](#) (I2S\_Type \*base, [sai\\_protocol\\_t](#) protocol, uint32\_t bits)  
*Set Rx word width.*

## 34.2.2 Data Structure Documentation

### 34.2.2.1 struct [sai\\_clock\\_setting\\_t](#)

#### Data Fields

- [sai\\_mclk\\_source\\_t](#) [mclk\\_src](#)  
*Master clock source.*
- [sai\\_bclk\\_source\\_t](#) [bclk\\_src](#)  
*Bit clock source.*
- uint32\_t [mclk\\_src\\_freq](#)  
*Master clock source frequency.*
- uint32\_t [mclk](#)  
*Master clock frequency.*
- uint32\_t [bclk](#)  
*Bit clock frequency.*

#### 34.2.2.1.0.58 Field Documentation

34.2.2.1.0.58.1 [sai\\_mclk\\_source\\_t](#) [sai\\_clock\\_setting\\_t::mclk\\_src](#)

34.2.2.1.0.58.2 [sai\\_bclk\\_source\\_t](#) [sai\\_clock\\_setting\\_t::bclk\\_src](#)

34.2.2.1.0.58.3 uint32\_t [sai\\_clock\\_setting\\_t::mclk\\_src\\_freq](#)

34.2.2.1.0.58.4 uint32\_t [sai\\_clock\\_setting\\_t::mclk](#)

34.2.2.1.0.58.5 uint32\_t [sai\\_clock\\_setting\\_t::bclk](#)

## 34.2.3 Enumeration Type Documentation

### 34.2.3.1 enum [sai\\_protocol\\_t](#)

Enumerator

*kSaiBusI2SLeft* Uses I2S left aligned format.

## SAI HAL driver

***kSaiBusI2SRight*** Uses I2S right aligned format.

***kSaiBusI2SType*** Uses I2S format.

***kSaiBusPCMA*** Uses I2S PCM A format.

***kSaiBusPCMB*** Uses I2S PCM B format.

***kSaiBusAC97*** Uses I2S AC97 format.

### 34.2.3.2 enum sai\_master\_slave\_t

Enumerator

***kSaiMaster*** Master mode.

***kSaiSlave*** Slave mode.

### 34.2.3.3 enum sai\_mono\_stereo\_t

Enumerator

***kSaiMono*** 1 channel in frame.

***kSaiStereo*** 2 channels in frame.

### 34.2.3.4 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.

### 34.2.3.5 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.



### 34.2.3.6 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.

### 34.2.3.7 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.

### 34.2.3.8 enum sai\_dma\_request\_t

Enumerator

- kSaiDmaReqFIFOWarning* FIFO warning caused by the DMA request.
- kSaiDmaReqFIFORequest* FIFO request caused by the DMA request.

### 34.2.3.9 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.
- kSaiStateFlagFIFORequest* FIFO request flag.
- kSaiStateFlagFIFOWarning* FIFO warning flag.
- kSaiStateFlagSoftReset* Software reset flag.
- kSaiStateFlagAll* All flags.

### 34.2.3.10 enum sai\_reset\_type\_t

Enumerator

- kSaiResetTypeSoftware* Software reset, reset the logic state.

## SAI HAL driver

*kSaiResetTypeFIFO* FIFO reset, reset the FIFO read and write pointer.  
*kSaiResetAll* All reset.

### 34.2.3.11 enum sai\_run\_mode\_t

Enumerator

*kSaiRunModeDebug* In debug mode.  
*kSaiRunModeStop* In stop mode.

## 34.2.4 Function Documentation

### 34.2.4.1 void SAI\_HAL\_TxInit ( I2S\_Type \* *base* )

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>base</i> | Register base address of SAI module. |
|-------------|--------------------------------------|

### 34.2.4.2 void SAI\_HAL\_RxInit ( I2S\_Type \* *base* )

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>base</i> | Register base address of SAI module. |
|-------------|--------------------------------------|

### 34.2.4.3 void SAI\_HAL\_TxSetProtocol ( I2S\_Type \* *base*, 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>base</i>     | Register base address of SAI module.                                        |
| <i>protocol</i> | The protocol selection. It can be I2S left aligned, I2S right aligned, etc. |

#### 34.2.4.4 void SAI\_HAL\_RxSetProtocol ( I2S\_Type \* *base*, 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>base</i>     | Register base address of SAI module.                                        |
| <i>protocol</i> | The protocol selection. It can be I2S left aligned, I2S right aligned, etc. |

#### 34.2.4.5 void SAI\_HAL\_TxSetMasterSlave ( I2S\_Type \* *base*, 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>base</i>              | Register base address of SAI module. |
| <i>master_slave_mode</i> | Mater or slave mode.                 |

#### 34.2.4.6 void SAI\_HAL\_RxSetMasterSlave ( I2S\_Type \* *base*, 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>base</i>              | Register base address of SAI module. |
| <i>master_slave_mode</i> | Mater or slave mode.                 |

## SAI HAL driver

**34.2.4.7 void SAI\_HAL\_TxClockSetup ( I2S\_Type \* *base*, sai\_clock\_setting\_t \* *clk\_config* )**

This function can sets the clock settings according to the configure structure. In this configuration setting structure, users can set clock source, clock source frequency, and frequency of master clock and bit clock. If bit clock source is master clock, the master clock frequency should equal to bit clock source frequency. If bit clock source is not master clock, then settings about master clock have no effect to the setting.

Parameters

|                   |                                               |
|-------------------|-----------------------------------------------|
| <i>base</i>       | Register base address of SAI module.          |
| <i>clk_config</i> | Pointer to sai clock configuration structure. |

**34.2.4.8 void SAI\_HAL\_RxClockSetup ( I2S\_Type \* *base*, sai\_clock\_setting\_t \* *clk\_config* )**

This function can sets the clock settings according to the configure structure. In this configuration setting structure, users can set clock source, clock source frequency, and frequency of master clock and bit clock. If bit clock source is master clock, the master clock frequency should equal to bit clock source frequency. If bit clock source is not master clock, then settings about master clock have no effect to the setting.

Parameters

|                   |                                               |
|-------------------|-----------------------------------------------|
| <i>base</i>       | Register base address of SAI module.          |
| <i>clk_config</i> | Pointer to sai clock configuration structure. |

**34.2.4.9 static void SAI\_HAL\_SetMclkSrc ( I2S\_Type \* *base*, 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>base</i>   | Register base address of SAI module. |
| <i>source</i> | Mater clock source                   |

```
34.2.4.10 static uint32_t SAI_HAL_GetMclkSrc (I2S_Type * base) [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.

## SAI HAL driver

### Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Register base address of SAI module. |
|-------------|--------------------------------------|

### Returns

Master clock source

**34.2.4.11 static void SAI\_HAL\_SetMclkDividerCmd ( I2S\_Type \* *base*, bool *enable* ) [inline], [static]**

This function enable or disable internal MCLK.

### Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | Register base address of SAI module.    |
| <i>enable</i> | True means enable, false means disable. |

**34.2.4.12 static void SAI\_HAL\_TxSetBclkSrc ( I2S\_Type \* *base*, 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>base</i>   | Register base address of SAI module. |
| <i>source</i> | Bit clock source.                    |

**34.2.4.13 static void SAI\_HAL\_RxSetBclkSrc ( I2S\_Type \* *base*, 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>base</i>   | Register base address of SAI module. |
| <i>source</i> | Bit clock source.                    |

**34.2.4.14 static uint32\_t SAI\_HAL\_TxGetBclkSrc ( I2S\_Type \* *base* ) [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>base</i> | Register base address of SAI module. |
|-------------|--------------------------------------|

Returns

Bit clock source.

**34.2.4.15 static uint32\_t SAI\_HAL\_RxGetBclkSrc ( I2S\_Type \* *base* ) [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>base</i> | Register base address of SAI module. |
|-------------|--------------------------------------|

Returns

Bit clock source.

**34.2.4.16 static void SAI\_HAL\_TxSetBclkDiv ( I2S\_Type \* *base*, 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.

## SAI HAL driver

### Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | Register base address of SAI module. |
| <i>divider</i> | The divide number of bit clock.      |

#### 34.2.4.17 static void SAI\_HAL\_RxSetBclkDiv ( I2S\_Type \* *base*, uint32\_t *divider* ) [inline], [static]

$blk = mclk / divider$ . At the same time,  $blk = sample\_rate * channel * bits$ . 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>base</i>    | Register base address of SAI module. |
| <i>divider</i> | The divide number of bit clock.      |

#### 34.2.4.18 static void SAI\_HAL\_TxSetBclkInputCmd ( I2S\_Type \* *base*, bool *enable* ) [inline], [static]

### Parameters

|                    |                                         |
|--------------------|-----------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.    |
| <i>enable</i>      | True means enable, false means disable. |

#### 34.2.4.19 static void SAI\_HAL\_RxSetBclkInputCmd ( I2S\_Type \* *base*, bool *enable* ) [inline], [static]

### Parameters

|                    |                                         |
|--------------------|-----------------------------------------|
| <i>saiBaseAddr</i> | Register base address of SAI module.    |
| <i>enable</i>      | True means enable, false means disable. |

#### 34.2.4.20 static void SAI\_HAL\_TxSetSwapBclkCmd ( I2S\_Type \* *base*, 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 (SAI\_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 clock, false means no swap. |

#### 34.2.4.21 **static void SAI\_HAL\_RxSetSwapBclkCmd ( I2S\_Type \* *base*, 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\_TX\_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 clock, false means no swap. |

#### 34.2.4.22 **void SAI\_HAL\_TxSetMonoStereo ( I2S\_Type \* *base*, sai\_mono\_stereo\_t *mono\_stereo* )**

Can be mono or stereo.

Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>base</i>        | Register base address of SAI module. |
| <i>mono_stereo</i> | Mono or stereo mode.                 |

#### 34.2.4.23 **void SAI\_HAL\_RxSetMonoStereo ( I2S\_Type \* *base*, sai\_mono\_stereo\_t *mono\_stereo* )**

Can be mono or stereo.

## SAI HAL driver

### Parameters

|                    |                                      |
|--------------------|--------------------------------------|
| <i>base</i>        | Register base address of SAI module. |
| <i>mono_stereo</i> | Mono or stereo mode.                 |

#### 34.2.4.24 void SAI\_HAL\_TxSetWordWidth ( I2S\_Type \* *base*, sai\_protocol\_t *protocol*, uint32\_t *bits* )

This interface is for i2s and PCM series protocol, it would set the width of first word and other word the same. At the same time, for i2s series protocol, it will set frame sync width the equal to the word width.

### Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>base</i>     | Register base address of SAI module. |
| <i>protocol</i> | Protocol used for tx now.            |
| <i>bits</i>     | Tx word width.                       |

#### 34.2.4.25 void SAI\_HAL\_RxSetWordWidth ( I2S\_Type \* *base*, sai\_protocol\_t *protocol*, uint32\_t *bits* )

This interface is for i2s and PCM series protocol, it would set the width of first word and other word the same. At the same time, for i2s series protocol, it will set frame sync width the equal to the word width.

### Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>base</i>     | Register base address of SAI module. |
| <i>protocol</i> | Protocol used for rx now.            |
| <i>bits</i>     | Rx word width.                       |

## 34.3 SAI Peripheral driver

### 34.3.1 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.

### 34.3.2 SAI Initialization

To initialize SAI, call the [SAI\\_DRV\\_TxInit\(\)](#) or [SAI\\_DRV\\_RxInit\(\)](#) functions and pass the SAI instance and SAI configuration structure parameters. The function opens the clock gate and initializes the SAI modules according to the structure information.

### 34.3.3 SAI Configuration

Configuration is implemented by the [SAI\\_DRV\\_TxConfigDataFormat\(\)](#) function. To use this function, transfer the audio data format to the SAI module.

### 34.3.4 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\\_SendDataInt\(\)](#) or the [SAI\\_DRV\\_SendDataDma\(\)](#) functions.
4. Stop Tx or Rx by calling the [SAI\\_DRV\\_TxStopModule\(\)](#) or the [SAI\\_DRV\\_RxStopModule\(\)](#) 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 configuration 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.watermark = 4;
tx_config.dma_source = kDmaRequestMux0I2S0Tx;

//SAI state, used for driver internal logic.
sai_state_t tx_state;

//Data format of audio data
```

## SAI Peripheral driver

```

audio_data_format_t format;
format.bits = 16;
format.sample_rate = 44100;
format.mclk = 384 * format->sample_rate;
format.mono_stereo = kSaiStereo;
//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_SendDataDma(instance, addr, len);

.....

//Stop Tx.
SAI_DRV_TxStopModule(instance);
//De-initialize 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](#)  
*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 receive 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 transmit module.*

  - `sai_status_t SAI_DRV_RxDeinit` (uint32\_t instance)

*De-initializes the SAI receive module.*

  - `sai_status_t SAI_DRV_TxConfigDataFormat` (uint32\_t instance, `sai_data_format_t *format`)

*Configures audio data format of the transmit.*

  - `sai_status_t SAI_DRV_RxConfigDataFormat` (uint32\_t instance, `sai_data_format_t *format`)

*Configures audio data format of the receive.*

  - `void SAI_DRV_TxStartModule` (uint32\_t instance)

*Starts the transmit transfer.*

  - `void SAI_DRV_RxStartModule` (uint32\_t instance)

*Starts the 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 transmit interrupt source.*

  - `static void SAI_DRV_RxSetIntCmd` (uint32\_t instance, bool enable)

*Enables or disables the receive interrupt source.*

  - `static void SAI_DRV_TxSetDmaCmd` (uint32\_t instance, bool enable)

*Enables or disables the transmit DMA source.*

  - `static void SAI_DRV_RxSetDmaCmd` (uint32\_t instance, bool enable)

*Enables or disables the receive interrupt source.*

  - `static uint32_t SAI_DRV_TxGetFifoAddr` (uint32\_t instance, uint32\_t fifo\_channel)

*Gets the transmit FIFO address of the data channel.*

  - `static uint32_t SAI_DRV_RxGetFifoAddr` (uint32\_t instance, uint32\_t fifo\_channel)

*Gets the receive FIFO address of the data channel.*

  - `uint32_t SAI_DRV_SendDataInt` (uint32\_t instance, uint8\_t \*addr, uint32\_t len)

*Sends data using interrupts.*

  - `uint32_t SAI_DRV_ReceiveDataInt` (uint32\_t instance, uint8\_t \*addr, uint32\_t len)

*Receives data a certain length using interrupt way.*

  - `uint32_t SAI_DRV_SendDataDma` (uint32\_t instance, uint8\_t \*addr, uint32\_t len)

*Sends data of a certain length using the DMA way.*

  - `uint32_t SAI_DRV_ReceiveDataDma` (uint32\_t instance, uint8\_t \*addr, uint32\_t len)

*Receives data using the DMA.*

  - `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 transmit interrupt handler.*

## SAI Peripheral driver

- void [SAI\\_DRV\\_RxIRQHandler](#) (uint32\_t instance)  
*Default SAI receive interrupt handler.*

### 34.3.5 Data Structure Documentation

#### 34.3.5.1 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\\_stereo\\_t mono\\_stereo](#)  
*How many word in a frame.*

#### 34.3.5.2 struct sai\_state\_t

Note: During the SAI execution, users should not free the state. Otherwise, the driver malfunctions.

#### 34.3.5.3 struct sai\_user\_config\_t

##### Data Fields

- [sai\\_mclk\\_source\\_t mclk\\_source](#)  
*Master clock source.*
- uint8\_t [channel](#)  
*Which FIFO is used to transfer.*
- [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.*
- uint32\_t [dma\\_source](#)  
*Dma request source.*

### 34.3.6 Function Documentation

#### 34.3.6.1 `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 transmit 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.

#### 34.3.6.2 `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 transmit 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.

#### 34.3.6.3 `void SAI_DRV_TxGetDefaultSetting ( sai_user_config_t * config )`

The default settings for SAI are:

- Audio protocol is I2S format
- Watermark is 4

## SAI Peripheral driver

- Use SAI0
- Channel is channel0
- SAI as master
- MCLK from system core clock
- Transmit is in an asynchronous mode

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>config</i> | Pointer of user configure structure. |
|---------------|--------------------------------------|

### 34.3.6.4 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 Receive is in synchronous way

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>config</i> | Pointer of user configure structure. |
|---------------|--------------------------------------|

### 34.3.6.5 sai\_status\_t SAI\_DRV\_TxDeinit ( uint32\_t instance )

This function closes the SAI transmit device. It does not close the entire SAI instance. It only closes the clock gate while both transmit and receive 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.

### 34.3.6.6 sai\_status\_t SAI\_DRV\_RxDeinit ( uint32\_t instance )

This function closes the SAI receive device. It does not close the entire SAI instance. It only closes the clock gate while both transmit and receive 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.

**34.3.6.7 `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.

**34.3.6.8 `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.

**34.3.6.9 `void SAI_DRV_TxStartModule ( uint32_t instance )`**

The function enables the interrupt/DMA request source and the transmit channel.

## SAI Peripheral driver

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>instance</i> | SAI module instance. |
|-----------------|----------------------|

### 34.3.6.10 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. |
|-----------------|----------------------------------------|

### 34.3.6.11 static void SAI\_DRV\_TxStopModule ( uint32\_t *instance* ) [inline], [static]

This function provides the method to pause writing data.

Parameters

|                 |                      |
|-----------------|----------------------|
| <i>instance</i> | SAI module instance. |
|-----------------|----------------------|

### 34.3.6.12 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. |
|-----------------|----------------------|

### 34.3.6.13 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. |
|---------------|---------------------------------------------------------------------------|

**34.3.6.14 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. |

**34.3.6.15 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. |

**34.3.6.16 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. |

**34.3.6.17 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

---

## SAI Peripheral driver

|                     |                                        |
|---------------------|----------------------------------------|
| <i>instance</i>     | SAI module instance of the SAI module. |
| <i>fifo_channel</i> | FIFO channel of SAI transmit.          |

Returns

Returns the address of the data channel FIFO.

**34.3.6.18** `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. |
| <i>fifo_channel</i> | FIFO channel of SAI receive.           |

Returns

Returns the address of the data channel FIFO.

**34.3.6.19** `uint32_t SAI_DRV_SendDataInt ( uint32_t instance, uint8_t * addr, uint32_t len )`

This function sends data to the transmit FIFO. This function starts the transfer, and, while finishing the transfer, calls the callback function registered by users. This function is an un-blocking function.

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 number of bytes which need to be sent.         |

Returns

Returns the length which was sent.

**34.3.6.20** `uint32_t SAI_DRV_ReceiveDataInt ( uint32_t instance, uint8_t * addr, uint32_t len )`

This function receives the data from the receive FIFO. This function starts the transfer, and, while finishing the transfer, calls the callback function registered by the user. This function is an un-blocking function.

Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>instance</i> | SAI module instance.                               |
| <i>addr</i>     | Address of the data which needs to be transferred. |
| <i>len</i>      | The number of bytes to receive.                    |

Returns

Returns the length received.

**34.3.6.21** `uint32_t SAI_DRV_SendDataDma ( uint32_t instance, uint8_t * addr, uint32_t len )`

This function sends the data to the transmit FIFO. This function starts the transfer, and, while finishing the transfer, calls the callback function registered by users. This function is an a-sync function.

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 number of bytes which need to be sent.         |

Returns

Returns the length which was sent.

**34.3.6.22** `uint32_t SAI_DRV_ReceiveDataDma ( uint32_t instance, uint8_t * addr, uint32_t len )`

This function receives the data from the receive FIFO. This function starts the transfer, and, while finishing the transfer, calls the callback function registered by the user. This function is an a-sync function.

Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>instance</i> | SAI module instance.                               |
| <i>addr</i>     | Address of the data which needs to be transferred. |

## SAI Peripheral driver

|            |                                 |
|------------|---------------------------------|
| <i>len</i> | The number of bytes to receive. |
|------------|---------------------------------|

Returns

Returns the length received.

### 34.3.6.23 void SAI\_DRV\_TxRegisterCallback ( uint32\_t *instance*, sai\_callback\_t *callback*, void \* *callback\_param* )

This function tells the 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.     |
| <i>callback_ -<br/>param</i> | The parameter of the callback function. |

### 34.3.6.24 void SAI\_DRV\_RxRegisterCallback ( uint32\_t *instance*, sai\_callback\_t *callback*, void \* *callback\_param* )

This function tells the 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_ -<br/>param</i> | The parameter of the callback function. |

### 34.3.6.25 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. |
|-----------------|----------------------|

#### 34.3.6.26 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 35

# Secured Digital Host Controller (SDHC)

### 35.1 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](#)

## SDHC HAL

### 35.2 SDHC HAL

#### 35.2.1 Overview

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

#### Data Structures

- struct [sdhc\\_hal\\_basic\\_info\\_t](#)  
*Data structure to get the basic information of SDHC. [More...](#)*
- struct [sdhc\\_hal\\_sdclk\\_config\\_t](#)  
*SD clock configuration to configure the clock of SD protocol unit. [More...](#)*
- struct [sdhc\\_mmcboot\\_param\\_t](#)  
*Data structure to configure the MMC boot feature. [More...](#)*
- struct [sdhc\\_hal\\_config\\_t](#)  
*Data structure to initialize the SDHC. [More...](#)*
- struct [sdhc\\_hal\\_cmd\\_req\\_t](#)  
*Command request structure. [More...](#)*

#### Macros

- #define [SDHC\\_HAL\\_ADMA1\\_ADDR\\_ALIGN](#) (4096)  
*SDHC ADMA address alignment size and length alignment size.*

#### Enumerations

- enum [sdhc\\_hal\\_mmcboot\\_t](#)  
*MMC card BOOT type.*
- enum [sdhc\\_hal\\_led\\_t](#)  
*Led control status.*
- enum [sdhc\\_hal\\_dtw\\_t](#)  
*Data transfer width.*
- enum [sdhc\\_hal\\_endian\\_t](#)  
*SDHC endian mode.*
- enum [sdhc\\_hal\\_dma\\_mode\\_t](#)  
*SDHC dma mode.*
- enum [sdhc\\_hal\\_curstat\\_type\\_t](#) {

```

kSdhcHalIsCmdInhibit,
kSdhcHalIsDataInhibit,
kSdhcHalIsDataLineActive,
kSdhcHalIsSdClockStable,
kSdhcHalIsIpgClockOff,
kSdhcHalIsSysClockOff,
kSdhcHalIsPeripheralClockOff,
kSdhcHalIsSdClkOff,
kSdhcHalIsWriteTransferActive,
kSdhcHalIsReadTransferActive,
kSdhcHalIsBuffWriteEnabled,
kSdhcHalIsBuffReadEnabled,
kSdhcHalIsCardInserted,
kSdhcHalIsCmdLineLevelHigh,
kSdhcHalGetDataLine0Level,
kSdhcHalGetDataLine1Level,
kSdhcHalGetDataLine2Level,
kSdhcHalGetDataLine3Level,
kSdhcHalGetDataLine4Level,
kSdhcHalGetDataLine5Level,
kSdhcHalGetDataLine6Level,
kSdhcHalGetDataLine7Level,
kSdhcHalGetCdTestLevel }

```

*Current sdhc status type.*

- enum `sdhc_hal_err_type_t` {  
`kAc12Err`,  
`kAdmaErr` }

*SDHC error type.*

## SDHC HAL FUNCTION

- void `SDHC_HAL_SendCmd` (SDHC\_Type \*base, const `sdhc_hal_cmd_req_t` \*cmdReq)  
*Sends command to card.*
- static void `SDHC_HAL_SetData` (SDHC\_Type \*base, uint32\_t data)  
*Fills the the data port.*
- static uint32\_t `SDHC_HAL_GetData` (SDHC\_Type \*base)  
*Retrieves the data from the data port.*
- bool `SDHC_HAL_GetCurState` (SDHC\_Type \*base, `sdhc_hal_curstat_type_t` stateType)  
*Gets current card's status.*
- static void `SDHC_HAL_SetDataTransferWidth` (SDHC\_Type \*base, `sdhc_hal_dtw_t` dtw)  
*Sets the data transfer width.*
- static void `SDHC_HAL_SetContinueRequest` (SDHC\_Type \*base)  
*Restarts a transaction which has stopped at the block gap.*
- void `SDHC_HAL_Config` (SDHC\_Type \*base, const `sdhc_hal_config_t` \*initConfig)  
*Initialize the SDHC according to the configuration user input.*
- void `SDHC_HAL_ConfigSdClock` (SDHC\_Type \*base, `sdhc_hal_sdclk_config_t` \*clkConfItms)

## SDHC HAL

- *Sets SDHC SD protol unit clock.*
- static uint32\_t [SDHC\\_HAL\\_GetIntFlags](#) (SDHC\_Type \*base)
- *Gets the current interrupt status.*
- static void [SDHC\\_HAL\\_ClearIntFlags](#) (SDHC\_Type \*base, uint32\_t mask)
- *Clears a specified interrupt status.*
- void [SDHC\\_HAL\\_GetAllErrStatus](#) (SDHC\_Type \*base, [sdhc\\_hal\\_err\\_type\\_t](#) errType, uint32\_t \*errFlags)
- *Get the error status of SDHC .*
- static void [SDHC\\_HAL\\_SetForceEventFlags](#) (SDHC\_Type \*base, uint32\_t mask)
- *Sets the force events according to the given mask.*
- static void [SDHC\\_HAL\\_SetAdmaAddress](#) (SDHC\_Type \*base, uint32\_t address)
- *Sets the ADMA address.*
- uint32\_t [SDHC\\_HAL\\_GetResponse](#) (SDHC\_Type \*base, uint32\_t index)
- *Gets the command response.*
- void [SDHC\\_HAL\\_SetIntSignal](#) (SDHC\_Type \*base, bool enable, uint32\_t mask)
- *Enables the specified interrupts.*
- void [SDHC\\_HAL\\_SetIntState](#) (SDHC\_Type \*base, bool enable, uint32\_t mask)
- *Enables the specified interrupt state.*
- uint32\_t [SDHC\\_HAL\\_Reset](#) (SDHC\_Type \*base, uint32\_t type, uint32\_t timeout)
- *Performs an SDHC reset.*
- uint32\_t [SDHC\\_HAL\\_InitCard](#) (SDHC\_Type \*base, uint32\_t timeout)
- *Sends 80 clocks to the card to initialize the card.*
- void [SDHC\\_HAL\\_Init](#) (SDHC\_Type \*base)
- *Initializes the SDHC HAL.*
- void [SDHC\\_HAL\\_GetBasicInfo](#) (SDHC\_Type \*base, [sdhc\\_hal\\_basic\\_info\\_t](#) \*basicInfo)
- *Get the capability of SDHC.*

## 35.2.2 Data Structure Documentation

### 35.2.2.1 struct sdhc\_hal\_basic\_info\_t

#### Data Fields

- uint8\_t [specVer](#)  
*Save the specification version.*
- uint8\_t [vendorVer](#)  
*Save the vendor version.*
- uint16\_t [maxBlkLen](#)  
*Save the max block length.*
- uint32\_t [capability](#)  
*The capability flags.*

**35.2.2.2 struct sdhc\_hal\_sdclk\_config\_t****35.2.2.3 struct sdhc\_mmcboot\_param\_t****Data Fields**

- uint32\_t [ackTimeout](#)  
*Sets the timeout value for the boot ACK.*
- [sdhc\\_hal\\_mmcboot\\_t mode](#)  
*Configures the boot mode.*
- uint32\_t [blockCount](#)  
*Configures the the block count for the boot.*

**35.2.2.3.0.59 Field Documentation****35.2.2.3.0.59.1 uint32\_t sdhc\_mmcboot\_param\_t::ackTimeout****35.2.2.3.0.59.2 sdhc\_hal\_mmcboot\_t sdhc\_mmcboot\_param\_t::mode****35.2.2.3.0.59.3 uint32\_t sdhc\_mmcboot\_param\_t::blockCount****35.2.2.4 struct sdhc\_hal\_config\_t****Data Fields**

- [sdhc\\_hal\\_led\\_t ledState](#)  
*Sets the LED state.*
- [sdhc\\_hal\\_endian\\_t endianMode](#)  
*Configures the endian mode.*
- [sdhc\\_hal\\_dma\\_mode\\_t dmaMode](#)  
*Sets the DMA mode.*
- uint8\_t [writeWatermarkLevel](#)  
*Sets the watermark for writing.*
- uint8\_t [readWatermarkLevel](#)  
*Sets the watermark for reading.*
- uint32\_t [enFlags](#)  
*Enable or disable corresponding feature.*
- [sdhc\\_mmcboot\\_param\\_t bootParams](#)  
*Configure read MMC card boot data feature.*

## SDHC HAL

### 35.2.2.4.0.60 Field Documentation

35.2.2.4.0.60.1 `sdhc_hal_led_t` `sdhc_hal_config_t::ledState`

35.2.2.4.0.60.2 `sdhc_hal_endian_t` `sdhc_hal_config_t::endianMode`

35.2.2.4.0.60.3 `sdhc_hal_dma_mode_t` `sdhc_hal_config_t::dmaMode`

35.2.2.4.0.60.4 `uint8_t` `sdhc_hal_config_t::writeWatermarkLevel`

35.2.2.4.0.60.5 `uint8_t` `sdhc_hal_config_t::readWatermarkLevel`

### 35.2.2.5 struct `sdhc_hal_cmd_req_t`

#### Data Fields

- `uint32_t` [dataBlkSize](#)  
*Cmd data Block size.*
- `uint32_t` [dataBlkCount](#)  
*Cmd data Block count.*
- `uint32_t` [arg](#)  
*Cmd argument.*
- `uint32_t` [index](#)  
*Cmd index.*
- `uint32_t` [flags](#)  
*Cmd Flags.*

## 35.2.3 Enumeration Type Documentation

### 35.2.3.1 enum `sdhc_hal_curstat_type_t`

Enumerator

***kSdhcHallsCmdInhibit*** Checks whether the command inhibit bit is set or not.

***kSdhcHallsDataInhibit*** Checks whether data inhibit bit is set or not.

***kSdhcHallsDataLineActive*** Checks whether data line is active.

***kSdhcHallsSdClockStable*** Checks whether the SD clock is stable or not.

***kSdhcHallsIpgClockOff*** Checks whether the IPG clock is off or not.

***kSdhcHallsSysClockOff*** Checks whether the system clock is off or not.

***kSdhcHallsPeripheralClockOff*** Checks whether the peripheral clock is off or not.

***kSdhcHallsSdClkOff*** Checks whether the SD clock is off or not.

***kSdhcHallsWriteTransferActive*** Checks whether the write transfer is active or not.

***kSdhcHallsReadTransferActive*** Checks whether the read transfer is active or not.

***kSdhcHallsBuffWriteEnabled*** Check whether the buffer write is enabled or not.

***kSdhcHallsBuffReadEnabled*** Checks whether the buffer read is enabled or not.

***kSdhcHallsCardInserted*** Checks whether the card is inserted or not.

***kSdhcHallsCmdLineLevelHigh*** Checks whether the command line signal is high or not.

- kSdhcHalGetDataLine0Level* Gets the data line 0 signal level or not.
- kSdhcHalGetDataLine1Level* Gets the data line 1 signal level or not.
- kSdhcHalGetDataLine2Level* Gets the data line 2 signal level or not.
- kSdhcHalGetDataLine3Level* Gets the data line 3 signal level or not.
- kSdhcHalGetDataLine4Level* Gets the data line 4 signal level or not.
- kSdhcHalGetDataLine5Level* Gets the data line 5 signal level or not.
- kSdhcHalGetDataLine6Level* Gets the data line 6 signal level or not.
- kSdhcHalGetDataLine7Level* Gets the data line 7 signal level or not.
- kSdhcHalGetCdTestLevel* Gets the card detect test level.

### 35.2.3.2 enum sdhc\_hal\_err\_type\_t

Enumerator

- kAc12Err* Auto CMD12 error.
- kAdmaErr* ADMA error.

## 35.2.4 Function Documentation

### 35.2.4.1 void SDHC\_HAL\_SendCmd ( SDHC\_Type \* *base*, const sdhc\_hal\_cmd\_req\_t \* *cmdReq* )

Parameters

|               |                           |
|---------------|---------------------------|
| <i>base</i>   | SDHC base address         |
| <i>cmdReq</i> | command request structure |

### 35.2.4.2 static void SDHC\_HAL\_SetData ( SDHC\_Type \* *base*, uint32\_t *data* ) [inline], [static]

Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | SDHC base address         |
| <i>data</i> | the data about to be sent |

### 35.2.4.3 static uint32\_t SDHC\_HAL\_GetData ( SDHC\_Type \* *base* ) [inline], [static]

## SDHC HAL

### Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | SDHC base address |
|-------------|-------------------|

### Returns

the data has been read

#### 35.2.4.4 **bool SDHC\_HAL\_GetCurState ( SDHC\_Type \* *base*, sdhc\_hal\_curstat\_type\_t *stateType* )**

### Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | SDHC base address |
|-------------|-------------------|

### Returns

the status if happened corresponding to stateType

- true: status flag has been set
- false: status flag has not been set

#### 35.2.4.5 **static void SDHC\_HAL\_SetDataTransferWidth ( SDHC\_Type \* *base*, sdhc\_hal\_dtw\_t *dtw* ) [inline], [static]**

### Parameters

|             |                     |
|-------------|---------------------|
| <i>base</i> | SDHC base address   |
| <i>dtw</i>  | data transfer width |

#### 35.2.4.6 **static void SDHC\_HAL\_SetContinueRequest ( SDHC\_Type \* *base* ) [inline], [static]**

### Parameters

---



|             |                   |
|-------------|-------------------|
| <i>base</i> | SDHC base address |
|-------------|-------------------|

**35.2.4.7 void SDHC\_HAL\_Config ( SDHC\_Type \* *base*, const sdhc\_hal\_config\_t \* *initConfig* )**

Parameters

|                   |                             |
|-------------------|-----------------------------|
| <i>base</i>       | SDHC base address           |
| <i>initConfig</i> | The configuration structure |

**35.2.4.8 void SDHC\_HAL\_ConfigSdClock ( SDHC\_Type \* *base*, sdhc\_hal\_sdclk\_config\_t \* *clkConfltms* )**

Parameters

|                    |                                                |
|--------------------|------------------------------------------------|
| <i>base</i>        | SDHC base address                              |
| <i>clkConfltms</i> | SDHC SD protol unit clock configuration items. |

**35.2.4.9 static uint32\_t SDHC\_HAL\_GetIntFlags ( SDHC\_Type \* *base* ) [inline], [static]**

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | SDHC base address |
|-------------|-------------------|

Returns

current interrupt flags

**35.2.4.10 static void SDHC\_HAL\_ClearIntFlags ( SDHC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

## SDHC HAL

### Parameters

|             |                                            |
|-------------|--------------------------------------------|
| <i>base</i> | SDHC base address                          |
| <i>mask</i> | to specify interrupts' flags to be cleared |

**35.2.4.11** void SDHC\_HAL\_GetAllErrStatus ( SDHC\_Type \* *base*, sdhc\_hal\_err\_type\_t *errType*, uint32\_t \* *errFlags* )

### Parameters

|                            |                        |
|----------------------------|------------------------|
| <i>base</i>                | SDHC base address      |
| <i>sdhc_hal_err_type_t</i> | the error type         |
| <i>errFlags</i>            | the result error flags |

**35.2.4.12** static void SDHC\_HAL\_SetForceEventFlags ( SDHC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

### Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <i>base</i> | SDHC base address                            |
| <i>mask</i> | to specify the force events' flags to be set |

**35.2.4.13** static void SDHC\_HAL\_SetAdmaAddress ( SDHC\_Type \* *base*, uint32\_t *address* ) [inline], [static]

### Parameters

|                |                   |
|----------------|-------------------|
| <i>base</i>    | SDHC base address |
| <i>address</i> | for ADMA transfer |

**35.2.4.14** uint32\_t SDHC\_HAL\_GetResponse ( SDHC\_Type \* *base*, uint32\_t *index* )

## Parameters

|              |                                         |
|--------------|-----------------------------------------|
| <i>base</i>  | SDHC base address                       |
| <i>index</i> | of response register, range from 0 to 3 |

**35.2.4.15** void SDHC\_HAL\_SetIntSignal ( SDHC\_Type \* *base*, bool *enable*, uint32\_t *mask* )

## Parameters

|               |                                       |
|---------------|---------------------------------------|
| <i>base</i>   | SDHC base address                     |
| <i>enable</i> | enable or disable                     |
| <i>mask</i>   | to specify interrupts to be isEnabled |

**35.2.4.16** void SDHC\_HAL\_SetIntState ( SDHC\_Type \* *base*, bool *enable*, uint32\_t *mask* )

## Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>base</i>   | SDHC base address                          |
| <i>enable</i> | enable or disable                          |
| <i>mask</i>   | to specify interrupts' state to be enabled |

**35.2.4.17** uint32\_t SDHC\_HAL\_Reset ( SDHC\_Type \* *base*, uint32\_t *type*, uint32\_t *timeout* )

## Parameters

|                |                   |
|----------------|-------------------|
| <i>base</i>    | SDHC base address |
| <i>type</i>    | the type of reset |
| <i>timeout</i> | timeout for reset |

## Returns

0 on success, else on error

**35.2.4.18** uint32\_t SDHC\_HAL\_InitCard ( SDHC\_Type \* *base*, uint32\_t *timeout* )

## SDHC HAL

### Parameters

|                |                             |
|----------------|-----------------------------|
| <i>base</i>    | SDHC base address           |
| <i>timeout</i> | timeout for initialize card |

### Returns

0 on success, else on error

### 35.2.4.19 void SDHC\_HAL\_Init ( SDHC\_Type \* *base* )

### Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | SDHC base address |
|-------------|-------------------|

### 35.2.4.20 void SDHC\_HAL\_GetBasicInfo ( SDHC\_Type \* *base*, sdhc\_hal\_basic\_info\_t \* *basicInfo* )

### Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | SDHC base address |
|-------------|-------------------|

## 35.3 SDHC Peripheral Driver

### 35.3.1 Overview

This section describes the programming interface of the SDHC Peripheral driver. The SDHC driver configures the secure digital host controller and provides an easy way to operate the SDHC module.

### 35.3.2 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 //
}
```

### 35.3.3 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 an 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);
req.argument |= value << (group * 4);
req.flags = FSL_SDHC_REQ_FLAGS_DATA_READ; // Set command flags //
req.respType = kSdhcRespTypeR1; // Set response type //

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 with a specific instance index.*
- [sdhc\\_status\\_t 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.*

### 35.3.4 Function Documentation

#### 35.3.4.1 [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

|                 |                                                                      |
|-----------------|----------------------------------------------------------------------|
| <i>instance</i> | the specific instance index                                          |
| <i>host</i>     | pointer to a place storing the <a href="#">sdhc_host_t</a> structure |

|               |                                   |
|---------------|-----------------------------------|
| <i>config</i> | initialization configuration data |
|---------------|-----------------------------------|

Returns

kStatus\_SDHC\_NoError if success

#### 35.3.4.2 sdhc\_status\_t SDHC\_DRV\_Shutdown ( uint32\_t instance )

Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>instance</i> | the instance index of host controller |
|-----------------|---------------------------------------|

Returns

kStatus\_SDHC\_NoError if success

#### 35.3.4.3 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

#### 35.3.4.4 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

#### 35.3.4.5 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

### 35.3.4.6 `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

### 35.3.4.7 `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 |
|-----------------|---------------------------------------|



## 35.4 SDHC Data Types

### 35.4.1 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_HostIsAlreadyInited,`
  - `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 }

```

## 35.4.2 Data Structure Documentation

### 35.4.2.1 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.*

### 35.4.2.2 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.*

### 35.4.2.3 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.*

### 35.4.2.4 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

### 35.4.3 Enumeration Type Documentation

#### 35.4.3.1 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\_HostIsAlreadyInitied* 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.

### 35.4.3.2 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.

### 35.4.3.3 enum sdhc\_power\_mode\_t

Enumerator

*kSdhcPowerModeRunning* SDHC is running.  
*kSdhcPowerModeSuspended* SDHC is suspended.  
*kSdhcPowerModeStopped* SDHC is stopped.

### 35.4.3.4 enum sdhc\_buswidth\_t

Enumerator

*kSdhcBusWidth1Bit* 1-bit bus width.  
*kSdhcBusWidth4Bit* 4-bit bus width.  
*kSdhcBusWidth8Bit* 8-bit bus width.

### 35.4.3.5 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

### 35.4.3.6 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.



## 35.5 SDHC Standard Definition

### 35.5.1 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\_PRSENT** (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\\_BLK\\_GAP\\_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)

- *SDHC NORMAL INTERRUPT CMD CRC ERROR BIT.*  
#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\\_ASYNC\\_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)

- *SDHC CAPABILITIES SLOT TYPE REMOVABLE.*
- #define **SDHC\_HCAP\_SLOT\_EMBEDDED** (0x1U)
- *SDHC CAPABILITIES SLOT TYPE EMBEDDED SLOT FOR ONE DEVICE.*
- #define **SDHC\_HCAP\_SLOT\_SHARED** (0x2U)
- *SDHC CAPABILITIES SLOT TYPE SHARED BUS SLOT.*
- #define **SDHC\_HOST\_CAPABILITIES\_1** (0x44U)
- *SDHC CAPABILITIES1 REG.*
- #define **SDHC\_HCAP\_SUPPORT\_SDR50** (0x00000001U)
- *SDHC CAPABILITIES1 SUPPORT SDR50.*
- #define **SDHC\_HCAP\_SUPPORT\_SDR104** (0x00000002U)
- *SDHC CAPABILITIES1 SUPPORT SDR104.*
- #define **SDHC\_HCAP\_SUPPORT\_DDR50** (0x00000004U)
- *SDHC CAPABILITIES1 SUPPORT DDR50.*
- #define **SDHC\_HCAP\_DRIVER\_TYPE\_A** (0x00000010U)
- *SDHC CAPABILITIES1 SUPPORT DRIVER TYPE A.*
- #define **SDHC\_HCAP\_DRIVER\_TYPE\_C** (0x00000020U)
- *SDHC CAPABILITIES1 SUPPORT DRIVER TYPE C.*
- #define **SDHC\_HCAP\_DRIVER\_TYPE\_D** (0x00000040U)
- *SDHC CAPABILITIES1 SUPPORT DRIVER TYPE D.*
- #define **SDHC\_HCAP\_RT\_TMCNT\_LSF** (8U)
- *SDHC CAPABILITIES1 TIMER COUNT FOR RETUNING SHIFT.*
- #define **SDHC\_HCAP\_RT\_TMCNT\_MASK** (0x00000F00U)
- *SDHC CAPABILITIES1 TIMER COUNT FOR RETUNING MASK.*
- #define **SDHC\_HCAP\_USE\_SDR50\_TUNE** (0x00002000U)
- *SDHC CAPABILITIES1 USE TUNING FOR SDR50.*
- #define **SDHC\_HCAP\_RT\_MODE\_LSF** (14U)
- *SDHC CAPABILITIES1 RETUNE MODE SHIFT.*
- #define **SDHC\_HCAP\_RT\_MODE\_MASK** (0x0000C000U)
- *SDHC CAPABILITIES1 RETUNE MODE MASK.*
- #define **SDHC\_HCAP\_CLK\_MUL\_LSF** (16U)
- *SDHC CAPABILITIES1 CLOCK MULTIPLIER SHIFT.*
- #define **SDHC\_HCAP\_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**

### **35.6 SDHC Card Related Standard Definition**

This chapter describes the card standard definition.

## **35.7 SDHC Card Definition**

This chapter describes the programming interface of the card data definition for FSL SD host controller.

---

## SDHC Card Driver

### 35.8 SDHC Card Driver

This chapter describes the programming interface of the card driver for FSL SD host controller.

## **35.9 SD Card SPI Data Definition**

The chapter describes the programming interface of the card data definition over SPI.

---

## SD Card SPI Driver

### 35.10 SD Card SPI Driver

The chapter describes the programming interface of the card driver over SPI.

## Chapter 36 LCD (SLCD)

### 36.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the segment LCD (SLCD) block of Kinetis devices.

#### Modules

- [SLCD HAL driver](#)
- [SLCD Peripheral driver](#)

## SLCD HAL driver

### 36.2 SLCD HAL driver

#### 36.2.1 Overview

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

#### Data Structures

- struct [slcd\\_blink\\_config\\_t](#)  
*This structure describes SLCD blink configuration. [More...](#)*
- struct [slcd\\_fault\\_detect\\_config\\_t](#)  
*This structure describes SLCD fault detection configuration. [More...](#)*
- struct [slcd\\_work\\_mode\\_t](#)  
*This union structure describes SLCD work mode configuration. [More...](#)*
- struct [slcd\\_clk\\_config\\_t](#)  
*This structure describes SLCD clock and frame config. [More...](#)*

#### Enumerations

- enum [slcd\\_load\\_adjust\\_t](#) {  
[kSLCDLowLoadOrFastestClkSrc](#) = 0U,  
[kSLCDLowLoadOrIntermediateClkSrc](#) = 1U,  
[kSLCDHighLoadOrIntermediateClkSrc](#) = 2U,  
[kSLCDHighLoadOrSlowestClkSrc](#) = 3U }  
*SLCD to handle different LCD glass capacitance.*
- enum [slcd\\_regulated\\_voltage\\_trim\\_t](#) {  
[kSLCDRegulatedVolatgeTrim00](#) = 0U,  
[kSLCDRegulatedVolatgeTrim01](#) = 1U,  
[kSLCDRegulatedVolatgeTrim02](#) = 2U,  
[kSLCDRegulatedVolatgeTrim03](#) = 3U,  
[kSLCDRegulatedVolatgeTrim04](#) = 4U,  
[kSLCDRegulatedVolatgeTrim05](#) = 5U,  
[kSLCDRegulatedVolatgeTrim06](#) = 6U,  
[kSLCDRegulatedVolatgeTrim07](#) = 7U,  
[kSLCDRegulatedVolatgeTrim08](#) = 8U,  
[kSLCDRegulatedVolatgeTrim09](#) = 9U,  
[kSLCDRegulatedVolatgeTrim10](#) = 10U,  
[kSLCDRegulatedVolatgeTrim11](#) = 11U,  
[kSLCDRegulatedVolatgeTrim12](#) = 12U,  
[kSLCDRegulatedVolatgeTrim13](#) = 13U,  
[kSLCDRegulatedVolatgeTrim14](#) = 14U,  
[kSLCDRegulatedVolatgeTrim15](#) = 15U }  
*adjust the regulated voltage to meet the desired contrast*
- enum [slcd\\_alt\\_clk\\_div\\_t](#) {



```

kSLCDAltClkDivFactor1 = 0U,
kSLCDAltClkDivFactor64 = 1U,
kSLCDAltClkDivFactor256 = 2U,
kSLCDAltClkDivFactor512 = 3U }

```

*SLCD alternate clock divider.*

- enum `slcd_clk_src_t` {  
kSLCDDefaultClk = 0U,  
kSLCDAAlternateClk = 1U }

*SLCD clock source.*

- enum `slcd_alt_clk_src_t` {  
kSLCDAltClkSrc1 = 0U,  
kSLCDAltClkSrc2 = 1U }

*SLCD alternate clock source.*

- enum `slcd_clk_prescaler_t`

*SLCD clock prescaler to generate frame frequency.*

- enum `slcd_duty_cyc_t` {  
kSLCD1DutyCyc = 0U,  
kSLCD1Div2DutyCyc = 1U,  
kSLCD1Div3DutyCyc = 2U,  
kSLCD1Div4DutyCyc = 3U,  
kSLCD1Div5DutyCyc = 4U,  
kSLCD1Div6DutyCyc = 5U,  
kSLCD1Div7DutyCyc = 6U,  
kSLCD1Div8DutyCyc = 7U }

*SLCD duty cycle.*

- enum `slcd_phase_index_t` {  
kSLCDPhaseA = 0U,  
kSLCDPhaseB = 1U,  
kSLCDPhaseC = 2U,  
kSLCDPhaseD = 3U,  
kSLCDPhaseE = 4U,  
kSLCDPhaseF = 5U,  
kSLCDPhaseG = 6U,  
kSLCDPhaseH = 7U }

*SLCD segment index.*

- enum `slcd_blink_mode_t` {  
kSLCDBlankDisplay = 0U,  
kSLCDAAltDisplay = 1U }

*SLCD blink mode.*

- enum `slcd_blink_rate_t` {

## SLCD HAL driver

```

kSLCDBlinkRate00 = 0U,
kSLCDBlinkRate01 = 1U,
kSLCDBlinkRate02 = 2U,
kSLCDBlinkRate03 = 3U,
kSLCDBlinkRate04 = 4U,
kSLCDBlinkRate05 = 5U,
kSLCDBlinkRate06 = 6U,
kSLCDBlinkRate07 = 7U }

```

*SLCD blink rate.*

- enum `slcd_power_supply_option_t` {

```

kSLCDPowerReserved0 = 0U,
kSLCDPowerReserved1 = 1U,
kSLCDPowerInternalVll3AndChargePump = 2U,
kSLCDPowerReserved2 = 3U,
kSLCDPowerExternalVll3AndResistorNetWork = 4U,
kSLCDPowerReserved3 = 5U,
kSLCDPowerExternalVll3AndChargePump = 6U,
kSLCDPowerVll1AndChargePump = 7U }

```

*This structure describes SLCD power supply configuration.*

- enum `slcd_fault_detect_clk_prescaler_t` {

```

kSLCDFaultSampleFreq1BusClk = 0U,
kSLCDFaultSampleFreq1Div2BusClk = 1U,
kSLCDFaultSampleFreq1Div4BusClk = 2U,
kSLCDFaultSampleFreq1Div8BusClk = 3U,
kSLCDFaultSampleFreq1Div16BusClk = 4U,
kSLCDFaultSampleFreq1Div32BusClk = 5U,
kSLCDFaultSampleFreq1Div64BusClk = 6U,
kSLCDFaultSampleFreq1Div128BusClk = 7U }

```

*SLCD fault detect clock prescaler.*

- enum `slcd_fault_detect_sample_win_width_t` {

```

kSLCDFaultDetectWinWidth4SampleClk = 0U,
kSLCDFaultDetectWinWidth8SampleClk = 1U,
kSLCDFaultDetectWinWidth16SampleClk = 2U,
kSLCDFaultDetectWinWidth32SampleClk = 3U,
kSLCDFaultDetectWinWidth64SampleClk = 4U,
kSLCDFaultDetectWinWidth128SampleClk = 5U,
kSLCDFaultDetectWinWidth256SampleClk = 6U,
kSLCDFaultDetectWinWidth512SampleClk = 7U }

```

*SLCD fault detect sample window width.*

- enum `slcd_int_type_t` {

```

kSLCDFrameFreqInt = 1U,
kSLCDFaultDetectionCompleteInt = 2U,
kSLCDEnableAllInt = 3U }

```

*This enum structure describes SLCD interrupt configuration.*

- enum `slcd_status_t` {

```

kStatus_SLCD_Success = 0x0U,
kStatus_SLCD_NullArgument = 0x1U,
kStatus_SLCD_Fail = 0x2U }
 SLCD status return codes.

```

## SLCD HAL.

### SLCD

- static void [SLCD\\_HAL\\_Enable](#) (LCD\_Type \*base)  
*Enables the SLCD module operation.*
- static void [SLCD\\_HAL\\_Disable](#) (LCD\_Type \*base)  
*Disables the SLCD module operation.*
- static void [SLCD\\_HAL\\_SetDutyCyc](#) (LCD\_Type \*base, [slcd\\_duty\\_cyc\\_t](#) dutyCyc)  
*Configure SLCD duty cycle.*
- static void [SLCD\\_HAL\\_SetBlinkCmd](#) (LCD\_Type \*base, bool enable)  
*Configures SLCD blink command.*
- static void [SLCD\\_HAL\\_SetAltDisplayModeCmd](#) (LCD\_Type \*base, bool enable)  
*Configure SLCD display mode.*
- static void [SLCD\\_HAL\\_SetBlankDisplayModeCmd](#) (LCD\_Type \*base, bool enable)  
*Configure SLCD blank mode.*
- static void [SLCD\\_HAL\\_SetFaultDetectCmd](#) (LCD\_Type \*base, bool enable)  
*Configure SLCD fault detection enable.*
- static bool [SLCD\\_HAL\\_GetFaultDetectCompleteFlag](#) (LCD\_Type \*base)  
*Return SLCD fault detect complete flag.*
- static void [SLCD\\_HAL\\_ClearFaultDetectCompleteFlag](#) (LCD\_Type \*base)  
*Clear SLCD fault detect complete flag.*
- static uint32\_t [SLCD\\_HAL\\_GetFaultDetectCounter](#) (LCD\_Type \*base)  
*Return SLCD fault detect counter.*
- static void [SLCD\\_HAL\\_SetPinsEnableCmd](#) (LCD\_Type \*base, uint8\_t highReg, uint32\_t data)  
*Configure ALL SLCD pins enabled states.*
- static void [SLCD\\_HAL\\_SetBackPlanePinsEnableCmd](#) (LCD\_Type \*base, uint8\_t highReg, uint32\_t data)  
*Configure ALL SLCD pins type in high back plane register.*
- static void [SLCD\\_HAL\\_SetPinWaveForm](#) (LCD\_Type \*base, uint8\_t pinIndex, uint8\_t waveForm)  
*Configure SLCD pin waveform phase.*
- static void [SLCD\\_HAL\\_SetBackPlanePhase](#) (LCD\_Type \*base, uint8\_t pinIndex, [slcd\\_phase\\_index\\_t](#) phase)  
*Configure SLCD pin waveform phase.*
- static void [SLCD\\_HAL\\_SetPinWaveFormPhaseCmd](#) (LCD\_Type \*base, uint8\_t pinIndex, [slcd\\_phase\\_index\\_t](#) phaseIndex, bool enable)  
*Configure SLCD pin waveform one phase.*
- void [SLCD\\_HAL\\_VoltageAndPowerSupplyConfig](#) (LCD\_Type \*base, [slcd\\_power\\_supply\\_option\\_t](#) powerSupply, [slcd\\_load\\_adjust\\_t](#) loadAdjust, [slcd\\_regulated\\_voltage\\_trim\\_t](#) trim)  
*Configure SLCD voltage and power supply.*
- void [SLCD\\_HAL\\_ClockConfig](#) (LCD\_Type \*base, const [slcd\\_clk\\_config\\_t](#) \*clkConfigPtr)  
*Configures SLCD clock.*
- void [SLCD\\_HAL\\_BlinkingModeConfig](#) (LCD\_Type \*base, const [slcd\\_blink\\_config\\_t](#) \*blinkConfigPtr)

## SLCD HAL driver

- Configures SLCD blinking mode.*

  - void [SLCD\\_HAL\\_FaultDetectionConfig](#) (LCD\_Type \*base, const [slcd\\_fault\\_detect\\_config\\_t](#) \*faultDetectConfigPtr)
- Configures SLCD fault detection.*

  - void [SLCD\\_HAL\\_SetLowPowerModeConfig](#) (LCD\_Type \*base, const [slcd\\_work\\_mode\\_t](#) \*workMode)
- Configure SLCD running status in doze mode.*

  - void [SLCD\\_HAL\\_SetIntCmd](#) (LCD\_Type \*base, [slcd\\_int\\_type\\_t](#) intType, bool enable)
- Configure SLCD frame frequency interrupt.*

  - void [SLCD\\_HAL\\_Init](#) (LCD\_Type \*base)

*Configure SLCD to a workable state.*

## 36.2.2 Data Structure Documentation

### 36.2.2.1 struct [slcd\\_blink\\_config\\_t](#)

#### Data Fields

- [slcd\\_blink\\_rate\\_t](#) `blinkRate`  
*SLCD blinking frequency configuration.*
- [slcd\\_blink\\_mode\\_t](#) `blinkMode`  
*Selects the slcd display mode during blinking period.*

### 36.2.2.2 struct [slcd\\_fault\\_detect\\_config\\_t](#)

#### Data Fields

- bool [faultDetectCompleteIntEnabled](#)  
*Fault detect complete interrupt enabled or disabled.*
- bool [faultDetectBackPlaneEnabled](#)  
*True means the type of pin id that fault detected is back plane otherwise front plane.*
- [uint8\\_t](#) [faultDetectPinIndex](#)  
*Fault detected pin id from 0 to 63.*
- [slcd\\_fault\\_detect\\_clk\\_prescaler\\_t](#) `prescaler`  
*Fault detected clock prescaler.*
- [slcd\\_fault\\_detect\\_sample\\_win\\_width\\_t](#) `winWidth`  
*Fault detected sample window width.*

### 36.2.2.3 struct [slcd\\_work\\_mode\\_t](#)

#### Data Fields

- bool [kSLCDEnableInDozeMode](#)  
*false means enabled in doze mode*
- bool [kSLCDEnableInStopMode](#)  
*false means enabled in stop mode*

### 36.2.2.4 struct slcd\_clk\_config\_t

## 36.2.3 Enumeration Type Documentation

### 36.2.3.1 enum slcd\_load\_adjust\_t

Enumerator

- kSLCDLowLoadOrFastestClkSrc* Adjust the resistor bias network in low load or selects fastest clock for charge pump.
- kSLCDLowLoadOrIntermediateClkSrc* Adjust the resistor bias network in low load or selects intermediate clock for charge pump.
- kSLCDHighLoadOrIntermediateClkSrc* Adjust the resistor bias network in high load or selects intermediate clock for charge pump.
- kSLCDHighLoadOrSlowestClkSrc* Adjust the resistor bias network in high load or selects slowest clock for charge pump.

### 36.2.3.2 enum slcd\_regulated\_voltage\_trim\_t

Enumerator

- kSLCDRegulatedVolatgeTrim00* Increase the voltage to 0.91V.
- kSLCDRegulatedVolatgeTrim01* Increase the voltage to 1.01V.
- kSLCDRegulatedVolatgeTrim02* Increase the voltage to 0.96V.
- kSLCDRegulatedVolatgeTrim03* Increase the voltage to 1.06V.
- kSLCDRegulatedVolatgeTrim04* Increase the voltage to 0.93V.
- kSLCDRegulatedVolatgeTrim05* Increase the voltage to 1.02V.
- kSLCDRegulatedVolatgeTrim06* Increase the voltage to 0.98V.
- kSLCDRegulatedVolatgeTrim07* Increase the voltage to 1.08V.
- kSLCDRegulatedVolatgeTrim08* Increase the voltage to 0.92V.
- kSLCDRegulatedVolatgeTrim09* Increase the voltage to 1.02V.
- kSLCDRegulatedVolatgeTrim10* Increase the voltage to 0.97V.
- kSLCDRegulatedVolatgeTrim11* Increase the voltage to 1.07V.
- kSLCDRegulatedVolatgeTrim12* Increase the voltage to 0.94V.
- kSLCDRegulatedVolatgeTrim13* Increase the voltage to 1.05V.
- kSLCDRegulatedVolatgeTrim14* Increase the voltage to 0.99V.
- kSLCDRegulatedVolatgeTrim15* Increase the voltage to 1.09V.

### 36.2.3.3 enum slcd\_alt\_clk\_div\_t

Enumerator

- kSLCDAltClkDivFactor1* No divide for alternate clock.
- kSLCDAltClkDivFactor64* Divide alternate clock with factor 64.

## SLCD HAL driver

*kSLCDAltClkDivFactor256* Divide alternate clock with factor 256.

*kSLCDAltClkDivFactor512* Divide alternate clock with factor 512.

### 36.2.3.4 enum slcd\_clk\_src\_t

Enumerator

*kSLCDDefaultClk* Select default clock as lcd clock source.

*kSLCDAlternateClk* Select alternate clock as lcd clock source.

### 36.2.3.5 enum slcd\_alt\_clk\_src\_t

Enumerator

*kSLCDAltClkSrc1* Select default alternate clock.

*kSLCDAltClkSrc2* Select alternate clock source 2.

### 36.2.3.6 enum slcd\_duty\_cyc\_t

Enumerator

*kSLCD1DutyCyc* Lcd use 1/1 duty cycle.

*kSLCD1Div2DutyCyc* Lcd use 1/2 duty cycle.

*kSLCD1Div3DutyCyc* Lcd use 1/3 duty cycle.

*kSLCD1Div4DutyCyc* Lcd use 1/4 duty cycle.

*kSLCD1Div5DutyCyc* Lcd use 1/5 duty cycle.

*kSLCD1Div6DutyCyc* Lcd use 1/6 duty cycle.

*kSLCD1Div7DutyCyc* Lcd use 1/7 duty cycle.

*kSLCD1Div8DutyCyc* Lcd use 1/8 duty cycle.

### 36.2.3.7 enum slcd\_phase\_index\_t

Enumerator

*kSLCDPhaseA* Lcd waveform phase A.

*kSLCDPhaseB* Lcd waveform phase B.

*kSLCDPhaseC* Lcd waveform phase C.

*kSLCDPhaseD* Lcd waveform phase D.

*kSLCDPhaseE* Lcd waveform phase E.

*kSLCDPhaseF* Lcd waveform phase F.

*kSLCDPhaseG* Lcd waveform phase G.

*kSLCDPhaseH* Lcd waveform phase H.

### 36.2.3.8 enum slcd\_blink\_mode\_t

Enumerator

- kSLCDBlinkDisplay* Display blank during the blink period.
- kSLCDAltDisplay* Display alternate display during the blink period if duty cycle is lower than 5.

### 36.2.3.9 enum slcd\_blink\_rate\_t

Enumerator

- kSLCDBlinkRate00* SLCD blink rate is LCD clock/ $((2^{12})$ )
- kSLCDBlinkRate01* SLCD blink rate is LCD clock/ $((2^{13})$ )
- kSLCDBlinkRate02* SLCD blink rate is LCD clock/ $((2^{14})$ )
- kSLCDBlinkRate03* SLCD blink rate is LCD clock/ $((2^{15})$ )
- kSLCDBlinkRate04* SLCD blink rate is LCD clock/ $((2^{16})$ )
- kSLCDBlinkRate05* SLCD blink rate is LCD clock/ $((2^{17})$ )
- kSLCDBlinkRate06* SLCD blink rate is LCD clock/ $((2^{18})$ )
- kSLCDBlinkRate07* SLCD blink rate is LCD clock/ $((2^{19})$ )

### 36.2.3.10 enum slcd\_power\_supply\_option\_t

Enumerator

- kSLCDPowerReserved0* Unused right now.
- kSLCDPowerReserved1* Unused right now.
- kSLCDPowerInternalVll3AndChargePump* VLL3 connected to VDD internally and charge pump is used to generate VLL1 and VLL2.
- kSLCDPowerReserved2* Unused right now.
- kSLCDPowerExternalVll3AndResistorNetWork* VLL3 is driven externally and resistor bias network is used to generate VLL1 and VLL2.
- kSLCDPowerReserved3* Unused right now.
- kSLCDPowerExternalVll3AndChargePump* VLL3 is driven externally and charge pump is used to generate VLL1 and VLL2.
- kSLCDPowerVll1AndChargePump* VIREG is connected to VLL1 internally and charge pump is used to generate VLL2 and VLL3.

### 36.2.3.11 enum slcd\_fault\_detect\_clk\_prescaler\_t

Enumerator

- kSLCDFaultSampleFreq1BusClk* Fault detect sample clock frequency is 1/1 bus clock.
- kSLCDFaultSampleFreq1Div2BusClk* Fault detect sample clock frequency is 1/2 bus clock.
- kSLCDFaultSampleFreq1Div4BusClk* Fault detect sample clock frequency is 1/4 bus clock.

## SLCD HAL driver

- kSLCDFaultSampleFreq1Div8BusClk* Fault detect sample clock frequency is 1/8 bus clock.
- kSLCDFaultSampleFreq1Div16BusClk* Fault detect sample clock frequency is 1/16 bus clock.
- kSLCDFaultSampleFreq1Div32BusClk* Fault detect sample clock frequency is 1/32 bus clock.
- kSLCDFaultSampleFreq1Div64BusClk* Fault detect sample clock frequency is 1/64 bus clock.
- kSLCDFaultSampleFreq1Div128BusClk* Fault detect sample clock frequency is 1/128 bus clock.

### 36.2.3.12 enum slcd\_fault\_detect\_sample\_win\_width\_t

Enumerator

- kSLCDFaultDetectWin Width4SampleClk* Sample window width is 4 sample clock cycles.
- kSLCDFaultDetectWin Width8SampleClk* Sample window width is 8 sample clock cycles.
- kSLCDFaultDetectWin Width16SampleClk* Sample window width is 16 sample clock cycles.
- kSLCDFaultDetectWin Width32SampleClk* Sample window width is 32 sample clock cycles.
- kSLCDFaultDetectWin Width64SampleClk* Sample window width is 64 sample clock cycles.
- kSLCDFaultDetectWin Width128SampleClk* Sample window width is 128 sample clock cycles.
- kSLCDFaultDetectWin Width256SampleClk* Sample window width is 256 sample clock cycles.
- kSLCDFaultDetectWin Width512SampleClk* Sample window width is 512 sample clock cycles.

### 36.2.3.13 enum slcd\_int\_type\_t

Enumerator

- kSLCDFrameFreqInt* SLCD frame frequency interrupt is enabled or disabled.
- kSLCDFaultDetectionCompleteInt* SLCD fault detection complete interrupt is enabled or disabled.
- kSLCDEnableAllInt* SLCD both frame frequency and fault detection interrupts are enabled or disabled.

### 36.2.3.14 enum slcd\_status\_t

Enumerator

- kStatus\_SLCD\_Success* Succeed.
- kStatus\_SLCD\_NullArgument* Argument is NULL.
- kStatus\_SLCD\_Fail* SLCD failed.

## 36.2.4 Function Documentation

**36.2.4.1** static void SLCD\_HAL\_Enable ( LCD\_Type \* base ) [inline], [static]



Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | Base address of SLCD peripheral instance. |
|-------------|-------------------------------------------|

**36.2.4.2 static void SLCD\_HAL\_Disable ( LCD\_Type \* *base* ) [inline], [static]**

Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | Base address of SLCD peripheral instance. |
|-------------|-------------------------------------------|

**36.2.4.3 static void SLCD\_HAL\_SetDutyCyc ( LCD\_Type \* *base*, slcd\_duty\_cyc\_t *dutyCyc* ) [inline], [static]**

Parameters

|                  |                                           |
|------------------|-------------------------------------------|
| <i>base</i>      | Base address of SLCD peripheral instance. |
| <i>dutyCycle</i> | Lcd duty cycle.                           |

**36.2.4.4 static void SLCD\_HAL\_SetBlinkCmd ( LCD\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>base</i>   | Base address of SLCD peripheral instance. |
| <i>enable</i> | Enable/Disable SLCD blinking mode.        |

**36.2.4.5 static void SLCD\_HAL\_SetAltDisplayModeCmd ( LCD\_Type \* *base*, bool *enable* ) [inline], [static]**

Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | Base address of SLCD peripheral instance.   |
| <i>enable</i> | Enable/Disable SLCD alternate display mode. |

**36.2.4.6 static void SLCD\_HAL\_SetBlankDisplayModeCmd ( LCD\_Type \* *base*, bool *enable* ) [inline], [static]**

## SLCD HAL driver

Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>base</i>   | Base address of SLCD peripheral instance. |
| <i>enable</i> | Enable/Disable SLCD blank mode.           |

**36.2.4.7 static void SLCD\_HAL\_SetFaultDetectCmd ( LCD\_Type \* *base*, bool *enable* )  
[inline], [static]**

Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>base</i>   | Base address of SLCD peripheral instance. |
| <i>enable</i> | Enable/Disable SLCD blank mode.           |

**36.2.4.8 static bool SLCD\_HAL\_GetFaultDetectCompleteFlag ( LCD\_Type \* *base* )  
[inline], [static]**

This function indicates that the fault detection is completed.

Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | Base address of SLCD peripheral instance. |
|-------------|-------------------------------------------|

**36.2.4.9 static void SLCD\_HAL\_ClearFaultDetectCompleteFlag ( LCD\_Type \* *base* )  
[inline], [static]**

This function clear fault detect complete flag.

Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | Base address of SLCD peripheral instance. |
|-------------|-------------------------------------------|

**36.2.4.10 static uint32\_t SLCD\_HAL\_GetFaultDetectCounter ( LCD\_Type \* *base* )  
[inline], [static]**

Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | Base address of SLCD peripheral instance. |
|-------------|-------------------------------------------|

**36.2.4.11** `static void SLCD_HAL_SetPinsEnableCmd ( LCD_Type * base, uint8_t highReg, uint32_t data ) [inline], [static]`

Parameters

|                |                                                                                |
|----------------|--------------------------------------------------------------------------------|
| <i>base</i>    | Base address of SLCD peripheral instance.                                      |
| <i>highReg</i> | 0 means low pin register, 1 means high pin register. SLCD pins enabled status. |

**36.2.4.12** `static void SLCD_HAL_SetBackPlanePinsEnableCmd ( LCD_Type * base, uint8_t highReg, uint32_t data ) [inline], [static]`

Parameters

|             |                                                                        |
|-------------|------------------------------------------------------------------------|
| <i>base</i> | Base address of SLCD peripheral instance.                              |
| <i>0</i>    | means low pin backpalne register, 1 means high pin backpalne register. |
| <i>data</i> | ALL SLCD back plane enabled status.                                    |

**36.2.4.13** `static void SLCD_HAL_SetPinWaveForm ( LCD_Type * base, uint8_t pinIndex, uint8_t waveForm ) [inline], [static]`

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>base</i>     | Base address of SLCD peripheral instance. |
| <i>pinIndex</i> | SLCD waveform number[0 63].               |
| <i>waveForm</i> | SLCD pin phase.                           |

**36.2.4.14** `static void SLCD_HAL_SetBackPlanePhase ( LCD_Type * base, uint8_t pinIndex, slcd_phase_index_t phase ) [inline], [static]`

## SLCD HAL driver

### Parameters

|                   |                                           |
|-------------------|-------------------------------------------|
| <i>base</i>       | Base address of SLCD peripheral instance. |
| <i>pinIndex</i>   | SLCD waveform number[0 63].               |
| <i>validPhase</i> | Configures SLCD back plane valid phase.   |

**36.2.4.15** `static void SLCD_HAL_SetPinWaveFormPhaseCmd ( LCD_Type * base,  
uint8_t pinIndex, slcd_phase_index_t phaseIndex, bool enable ) [inline],  
[static]`

### Parameters

|                   |                                              |
|-------------------|----------------------------------------------|
| <i>base</i>       | Base address of SLCD peripheral instance.    |
| <i>pinIndex</i>   | SLCD waveform number[0 63].                  |
| <i>phaseIndex</i> | SLCD pin phase.                              |
| <i>enable</i>     | Enable/Disable the specific phase for a pin. |

**36.2.4.16** `void SLCD_HAL_VoltageAndPowerSupplyConfig ( LCD_Type * base,  
slcd_power_supply_option_t powerSupply, slcd_load_adjust_t loadAdjust,  
slcd_regulated_voltage_trim_t trim )`

### Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>base</i>        | Base address of SLCD peripheral instance. |
| <i>powerSupply</i> | SLCD power supply configuration.          |
| <i>loadAdjust</i>  | SLCD load adjust configuration.           |

**36.2.4.17** `void SLCD_HAL_ClockConfig ( LCD_Type * base, const slcd_clk_config_t *  
clkConfigPtr )`

### Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>base</i>         | Base address of SLCD peripheral instance. |
| <i>clkSrc</i>       | SLCD clock source.                        |
| <i>altClkDiv</i>    | Alternating clock divider.                |
| <i>clkPrescaler</i> | Frame frequency clock prescaler.          |

36.2.4.18 void SLCD\_HAL\_BlinkingModeConfig ( LCD\_Type \* *base*, const  
slcd\_blink\_config\_t \* *blinkConfigPtr* )

## SLCD HAL driver

### Parameters

|                       |                                            |
|-----------------------|--------------------------------------------|
| <i>base</i>           | Base address of SLCD peripheral instance.  |
| <i>blinkConfigPtr</i> | Pointer of configuring SLCD blinking mode. |

### 36.2.4.19 void SLCD\_HAL\_FaultDetectionConfig ( LCD\_Type \* *base*, const slcd\_fault\_detect\_config\_t \* *faultDetectConfigPtr* )

### Parameters

|                              |                                           |
|------------------------------|-------------------------------------------|
| <i>base</i>                  | Base address of SLCD peripheral instance. |
| <i>faultDetect-ConfigPtr</i> | SLCD fault detection configuration.       |

### 36.2.4.20 void SLCD\_HAL\_SetLowPowerModeConfig ( LCD\_Type \* *base*, const slcd\_work\_mode\_t \* *workMode* )

### Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>base</i>     | Base address of SLCD peripheral instance.   |
| <i>workMode</i> | Configure SLCD work mode in doze/stop mode. |

### 36.2.4.21 void SLCD\_HAL\_SetIntCmd ( LCD\_Type \* *base*, slcd\_int\_type\_t *intType*, bool *enable* )

### Parameters

|                |                                           |
|----------------|-------------------------------------------|
| <i>base</i>    | Base address of SLCD peripheral instance. |
| <i>intType</i> | SLCD interrupt Configuration.             |
| <i>enable</i>  | Enable or Disable interrupt.              |

### 36.2.4.22 void SLCD\_HAL\_Init ( LCD\_Type \* *base* )

## Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | Base address of SLCD peripheral instance. |
|-------------|-------------------------------------------|

## SLCD Peripheral driver

### 36.3 SLCD Peripheral driver

#### 36.3.1 Overview

This section describes the programming interface of the SLCD Peripheral driver. The SLCD driver configures SLCD.

#### 36.3.2 Initialization

To initialize the SLCD module, call the [SLCD\\_DRV\\_Init\(\)](#) function and provide the user configuration data structure. This function sets the configuration of the SLCD module automatically and enables the SLCD module.

Note that if the configuration does not enable the SLCD, the application enables it after the configurations are completed.

This is example code to configure the SLCD driver: [slcd\\_work\\_mode\\_t](#) lowPowerMode = { .kSLCD-EnableInDozeMode = false, .kSLCDEnableInStopMode = false, }; [slcd\\_user\\_config\\_t](#) init = { .clkSrc = kSLCDDefaultClk, .alterClkDiv = kSLCDAltClkDivFactor1, .powerSupply = kSLCDPowerInternalV113-AndChargePump, .loadAdjust = kSLCDHighLoadOrSlowestClkSrc, .dutyCyc = kSLCD1Div8DutyCyc, .frameFreqDiv = kSLCDClkPrescaler02, .trim = kSLCDRegulatedVolatgeTrim08, .slcdIntEnabled = true, .workMode = lowPowerMode, }; [slcd\\_pins\\_config\\_t](#) pinsConfig = { .slcdLowPinsEnabled = 0x9FFFFE00, .slcdHighPinsEnabled = 0x0000083F, .slcdBackPlaneLowPinsEnabled = 0x0001FE00, .slcdBackPlaneHighPinsEnabled = 0x00000000, }; [slcd\\_fault\\_detect\\_config\\_t](#) faultDetectConfig = { .faultDetectCompleteIntEnabled = true, .faultDetectBackPlaneEnabled = true, .faultDetectPinIndex = 0, .prescaler = kSLCDFaultSampleFreq1BusClk, .winWidth = kSLCDFaultDetectWinWidth4SampleClk, }; [slcd\\_blink\\_config\\_t](#) blinkConfig = { .blinkRate = kSLCDBlinkRate01, .blinkMode = kSLCDAltDisplay, }; [configure\\_lcd\\_pins\(0\)](#);

```
SLCD_DRV_Init(0, &init);
```

```
SLCD_DRV_SetAllPinsConfig(0, &pinsConfig);
```

#### Data Structures

- struct [slcd\\_pins\\_config\\_t](#)  
*This structure describes SLCD pin configuration. [More...](#)*
- struct [slcd\\_user\\_config\\_t](#)  
*This structure describes the programming interface of the for SLCD initialization. [More...](#)*

#### Variables

- LCD\_Type \*const [g\\_slcdBase](#) [LCD\_INSTANCE\_COUNT]  
*Table of base addresses for SLCD instances.*
- const IRQn\_Type [g\\_slcdIrqId](#) [LCD\_INSTANCE\_COUNT]  
*Table to save SLCD IRQ enumeration numbers defined in the CMSIS header file.*



**SLCD Driver**

- **slcd\_status\_t SLCD\_DRV\_Init** (uint32\_t instance, const **slcd\_user\_config\_t** \*userConfigPtr)  
*Initializes the SLCD driver.*
- void **SLCD\_DRV\_Deinit** (uint32\_t instance)  
*De-initializes the SLCD driver.*
- void **SLCD\_DRV\_Start** (uint32\_t instance)  
*Starts the SLCD driver.*
- void **SLCD\_DRV\_Stop** (uint32\_t instance)  
*Stops the SLCD driver.*
- void **SLCD\_DRV\_SetIntCmd** (uint32\_t instance, **slcd\_int\_type\_t** intType, bool enable)  
*Configures the SLCD interrupt.*
- **slcd\_status\_t SLCD\_DRV\_StartFaultDetection** (uint32\_t instance, const **slcd\_fault\_detect\_config\_t** \*faultDetectConfigPtr)  
*Starts the SLCD fault detection.*
- bool **SLCD\_DRV\_CheckFaultDetectCompleteFlag** (uint32\_t instance)  
*Returns the fault detection configuration.*
- void **SLCD\_DRV\_ClearFaultDetectCompleteFlag** (uint32\_t instance)  
*Clears the fault detection complete flag.*
- uint8\_t **SLCD\_DRV\_GetFaultDetectCounter** (uint32\_t instance)  
*Returns the fault detection counter.*
- bool **SLCD\_DRV\_CheckFrameFrequencyIntFlag** (uint32\_t instance)  
*Checks the SLCD Frame frequency interrupt enabled or disabled.*
- void **SLCD\_DRV\_ClearFrameFrequencyIntFlag** (uint32\_t instance)  
*Clears the SLCD Frame frequency interrupt flag.*
- **slcd\_status\_t SLCD\_DRV\_StartBlinkingMode** (uint32\_t instance, const **slcd\_blink\_config\_t** \*blinkConfigPtr)  
*Starts the SLCD blinking mode.*
- void **SLCD\_DRV\_StopBlinkingMode** (uint32\_t instance)  
*Stops the SLCD blinking mode.*
- void **SLCD\_DRV\_SetAltDisplayModeCmd** (uint32\_t instance, bool enable)  
*Starts/stops SLCD alternating display mode.*
- void **SLCD\_DRV\_SetBlankDisplayModeCmd** (uint32\_t instance, bool enable)  
*Starts/stops SLCD blank display mode.*
- **slcd\_status\_t SLCD\_DRV\_SetAllPinsConfig** (uint32\_t instance, const **slcd\_pins\_config\_t** \*pinsConfigPtr)  
*Sets all SLCD pins configurations.*
- void **SLCD\_DRV\_SetPinWaveForm** (uint32\_t instance, uint8\_t pinIndex, uint8\_t waveForm)  
*Sets all SLCD pin waveforms.*
- void **SLCD\_DRV\_SetBackPlanePhase** (uint32\_t instance, uint8\_t pinIndex, **slcd\_phase\_index\_t** phase)  
*Sets the SLCD back plane phase.*
- void **SLCD\_DRV\_SetSegmentEnableCmd** (uint32\_t instance, uint8\_t pinIndex, **slcd\_phase\_index\_t** phaseIndex, bool enable)  
*Configures an SLCD pin segment.*

## SLCD Peripheral driver

### 36.3.3 Data Structure Documentation

#### 36.3.3.1 struct slcd\_pins\_config\_t

This structure is used when calling the SLCD\_DRV\_Init function.

##### Data Fields

- [uint32\\_t slcdLowPinsEnabled](#)  
*low pins from 0 to 31 and if one bit is set means the corresponding pin is enabled*
- [uint32\\_t slcdHighPinsEnabled](#)  
*high pins from 32 to 63 and if one bit is set means the corresponding pin is enabled*
- [uint32\\_t slcdBackPlaneLowPinsEnabled](#)  
*low pins from 0 to 31 and if one bit is set means the corresponding pin is configured as back plane*
- [uint32\\_t slcdBackPlaneHighPinsEnabled](#)  
*high pins from 32 to 63 and if the bit is set means the corresponding pin is configured as back plane*

#### 36.3.3.2 struct slcd\_user\_config\_t

This structure is used when calling the SLCD\_DRV\_Init function.

##### Data Fields

- [slcd\\_clk\\_config\\_t clkConfig](#)  
*Configures SLCD clock and frame frequency.*
- [slcd\\_power\\_supply\\_option\\_t powerSupply](#)  
*Configures SLCD power supply.*
- [slcd\\_load\\_adjust\\_t loadAdjust](#)  
*Configures charge pump or resistor bias to handle different LCD glass capacitance.*
- [slcd\\_duty\\_cyc\\_t dutyCyc](#)  
*Selects the duty cycle of the LCD controller driver.*
- [slcd\\_regulated\\_voltage\\_trim\\_t trim](#)  
*Adjusts SLCD contrast.*
- [bool slcdIntEnable](#)  
*SLCD interrupt enable or disable.*
- [slcd\\_work\\_mode\\_t workMode](#)  
*Configures SLCD low power work mode.*

### 36.3.4 Function Documentation

#### 36.3.4.1 `slcd_status_t SLCD_DRV_Init ( uint32_t instance, const slcd_user_config_t * userConfigPtr )`

Parameters

|                      |                                                                                            |
|----------------------|--------------------------------------------------------------------------------------------|
| <i>instance</i>      | The SLCD peripheral instance number.                                                       |
| <i>userConfigPtr</i> | The pointer to the SLCD user configure structure, see <a href="#">slcd_user_config_t</a> . |

Returns

kStatus\_SLCD\_Success means success. Otherwise, means failure.

**36.3.4.2 void SLCD\_DRV\_Deinit ( uint32\_t *instance* )**

Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>instance</i> | The SLCD peripheral instance number. |
|-----------------|--------------------------------------|

Returns

kStatus\_SLCD\_Success means success. Otherwise, means failure.

**36.3.4.3 void SLCD\_DRV\_Start ( uint32\_t *instance* )**

Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>instance</i> | The SLCD peripheral instance number. |
|-----------------|--------------------------------------|

**36.3.4.4 void SLCD\_DRV\_Stop ( uint32\_t *instance* )**

Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>instance</i> | The SLCD peripheral instance number. |
|-----------------|--------------------------------------|

**36.3.4.5 void SLCD\_DRV\_SetIntCmd ( uint32\_t *instance*, slcd\_int\_type\_t *intType*, bool *enable* )**

## SLCD Peripheral driver

### Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>instance</i> | The SLCD peripheral instance number. |
| <i>intType</i>  | interrupt type.                      |
| <i>enable</i>   | Enables or Disables interrupt.       |

### 36.3.4.6 `slcd_status_t SLCD_DRV_StartFaultDetection ( uint32_t instance, const slcd_fault_detect_config_t * faultDectectConfigPtr )`

### Parameters

|                              |                                                |
|------------------------------|------------------------------------------------|
| <i>instance</i>              | The SLCD peripheral instance number.           |
| <i>faultDetect-ConfigPtr</i> | The pointer to SLCD configure fault detection. |

### Returns

kStatus\_SLCD\_Success means success. Otherwise, means failure.

### 36.3.4.7 `bool SLCD_DRV_CheckFaultDetectCompleteFlag ( uint32_t instance )`

### Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>instance</i> | The SLCD peripheral instance number. |
|-----------------|--------------------------------------|

### Returns

true Fault detection is complete.  
false Fault detection is not complete.

### 36.3.4.8 `void SLCD_DRV_ClearFaultDetectCompleteFlag ( uint32_t instance )`

### Parameters

---

|                 |                                      |
|-----------------|--------------------------------------|
| <i>instance</i> | The SLCD peripheral instance number. |
|-----------------|--------------------------------------|

#### 36.3.4.9 uint8\_t SLCD\_DRV\_GetFaultDetectCounter ( uint32\_t *instance* )

Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>instance</i> | The SLCD peripheral instance number. |
|-----------------|--------------------------------------|

#### 36.3.4.10 bool SLCD\_DRV\_CheckFrameFrequencyIntFlag ( uint32\_t *instance* )

Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>instance</i> | The SLCD peripheral instance number. |
|-----------------|--------------------------------------|

Returns

- True This event causes an interrupt request.
- False No interrupt request is generated by this event.

#### 36.3.4.11 void SLCD\_DRV\_ClearFrameFrequencyIntFlag ( uint32\_t *instance* )

Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>instance</i> | The SLCD peripheral instance number. |
|-----------------|--------------------------------------|

#### 36.3.4.12 slcd\_status\_t SLCD\_DRV\_StartBlinkingMode ( uint32\_t *instance*, const slcd\_blink\_config\_t \* *blinkConfigPtr* )

Parameters

|                       |                                                         |
|-----------------------|---------------------------------------------------------|
| <i>instance</i>       | The SLCD peripheral instance number.                    |
| <i>blinkConfigPtr</i> | The pointer used to configure SLCD blink configuration. |

Returns

- kStatus\_SLCD\_Success means success. Otherwise, means failure.

#### 36.3.4.13 void SLCD\_DRV\_StopBlinkingMode ( uint32\_t *instance* )

## SLCD Peripheral driver

Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>instance</i> | The SLCD peripheral instance number. |
|-----------------|--------------------------------------|

### 36.3.4.14 void SLCD\_DRV\_SetAltDisplayModeCmd ( uint32\_t *instance*, bool *enable* )

Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>instance</i> | The SLCD peripheral instance number.          |
| <i>enable</i>   | Disable/Enable SLCD alternating display mode. |

### 36.3.4.15 void SLCD\_DRV\_SetBlankDisplayModeCmd ( uint32\_t *instance*, bool *enable* )

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>instance</i> | The SLCD peripheral instance number.    |
| <i>enable</i>   | Disable/Enable SLCD blank display mode. |

### 36.3.4.16 slcd\_status\_t SLCD\_DRV\_SetAllPinsConfig ( uint32\_t *instance*, const slcd\_pins\_config\_t \* *pinsConfigPtr* )

Parameters

|                      |                                          |
|----------------------|------------------------------------------|
| <i>instance</i>      | The SLCD peripheral instance number.     |
| <i>pinsConfigPtr</i> | The pointer used to configure SLCD pins. |

Returns

kStatus\_SLCD\_Success means success. Otherwise, means failure.

### 36.3.4.17 void SLCD\_DRV\_SetPinWaveForm ( uint32\_t *instance*, uint8\_t *pinIndex*, uint8\_t *waveForm* )

Parameters

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>instance</i> | The SLCD peripheral instance number.              |
| <i>pinIndex</i> | The configured SLCD pin index form 0 to 63.       |
| <i>waveForm</i> | The waveForm used to configure the pin wave form. |

**36.3.4.18 void SLCD\_DRV\_SetBackPlanePhase ( uint32\_t *instance*, uint8\_t *pinIndex*, slcd\_phase\_index\_t *phase* )**

Parameters

|                 |                                                   |
|-----------------|---------------------------------------------------|
| <i>instance</i> | The SLCD peripheral instance number.              |
| <i>pinIndex</i> | The configured SLCD pin index form 0 to 63.       |
| <i>waveForm</i> | The waveForm used to configure the pin wave form. |

**36.3.4.19 void SLCD\_DRV\_SetSegmentEnableCmd ( uint32\_t *instance*, uint8\_t *pinIndex*, slcd\_phase\_index\_t *phaseIndex*, bool *enable* )**

Parameters

|                   |                                      |
|-------------------|--------------------------------------|
| <i>instance</i>   | The SLCD peripheral instance number. |
| <i>pinIndex</i>   | The specific pin index from 0 to 63. |
| <i>phaseIndex</i> | The configured segment from 0 to 7.  |
| <i>enable</i>     | Enable/Disable the segment.          |

**36.3.5 Variable Documentation**

**36.3.5.1 LCD\_Type\* const g\_slcdBase[LCD\_INSTANCE\_COUNT]**

**36.3.5.2 const IRQn\_Type g\_slcdIrqId[LCD\_INSTANCE\_COUNT]**





## Chapter 37

# Serial Peripheral Interface (SPI)

### 37.1 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](#)

## SPI HAL driver

### 37.2 SPI HAL driver

#### 37.2.1 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_interrupt_source_t` {  
`kSpiRxFullAndModfInt` = 1,  
`kSpiTxEmptyInt` = 2,  
`kSpiMatchInt` = 3 }
- SPI interrupt sources.*

  - enum `spi_int_status_flag_t` {  
`kSpiRxBufferFullFlag` = SPI\_S\_SPRF\_SHIFT,  
`kSpiMatchFlag` = SPI\_S\_SPMF\_SHIFT,  
`kSpiTxBufferEmptyFlag` = SPI\_S\_SPTEF\_SHIFT,  
`kSpiModeFaultFlag` = SPI\_S\_MODF\_SHIFT }
- SPI interrupt status flags.*

  - enum `spi_fifo_interrupt_source_t` {  
`kSpiRxFifoNearFullInt` = 1,  
`kSpiTxFifoNearEmptyInt` = 2 }
- SPI FIFO interrupt sources.*

  - enum `spi_w1c_interrupt_t` {  
`kSpiRxFifoFullClearInt` = 0,  
`kSpiTxFifoEmptyClearInt` = 1,  
`kSpiRxNearFullClearInt` = 2,  
`kSpiTxNearEmptyClearInt` = 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` {

## SPI HAL driver

```

kSpiNoFifoError = 0,
kSpiRxfof = 1,
kSpiTxfof = 2,
kSpiRxfofTxfof = 3,
kSpiRxferr = 4,
kSpiRxfofRxferr = 5,
kSpiTxfofRxferr = 6,
kSpiRxfofTxfofRxferr = 7,
kSpiTxferr = 8,
kSpiRxfofTxferr = 9,
kSpiTxfofTxferr = 10,
kSpiRxfofTxfofTxferr = 11,
kSpiRxferrTxferr = 12,
kSpiRxfofRxferrTxferr = 13,
kSpiTxfofRxferrTxferr = 14,
kSpiRxfofTxfofRxferrTxferr = 15 }
 SPI error flags.

```

## Configuration

- void [SPI\\_HAL\\_Init](#) (SPI\_Type \*base)  
*Restores the SPI to reset configuration.*
- static void [SPI\\_HAL\\_Enable](#) (SPI\_Type \*base)  
*Enables the SPI peripheral.*
- static void [SPI\\_HAL\\_Disable](#) (SPI\_Type \*base)  
*Disables the SPI peripheral.*
- uint32\_t [SPI\\_HAL\\_SetBaud](#) (SPI\_Type \*base, uint32\_t bitsPerSec, uint32\_t sourceClockInHz)  
*Sets the SPI baud rate in bits per second.*
- static void [SPI\\_HAL\\_SetBaudDivisors](#) (SPI\_Type \*base, uint32\_t prescaleDivisor, uint32\_t rate-Divisor)  
*Configures the baud rate divisors manually.*
- static void [SPI\\_HAL\\_SetMasterSlave](#) (SPI\_Type \*base, [spi\\_master\\_slave\\_mode\\_t](#) mode)  
*Configures the SPI for master or slave.*
- static bool [SPI\\_HAL\\_IsMaster](#) (SPI\_Type \*base)  
*Returns whether the SPI module is in master mode.*
- void [SPI\\_HAL\\_SetSlaveSelectOutputMode](#) (SPI\_Type \*base, [spi\\_ss\\_output\\_mode\\_t](#) mode)  
*Sets how the slave select output operates.*
- void [SPI\\_HAL\\_SetDataFormat](#) (SPI\_Type \*base, [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](#) (SPI\_Type \*base)  
*Gets the SPI data register address for DMA operation.*
- void [SPI\\_HAL\\_SetPinMode](#) (SPI\_Type \*base, [spi\\_pin\\_mode\\_t](#) mode)  
*Sets the SPI pin mode.*

## Interrupts

- void [SPI\\_HAL\\_SetIntMode](#) (SPI\_Type \*base, spi\_interrupt\_source\_t interrupt, bool enable)  
*Enables or disables the SPI interrupts.*
- static void [SPI\\_HAL\\_SetReceiveAndFaultIntCmd](#) (SPI\_Type \*base, bool enable)  
*Enables or disables the SPI receive buffer/FIFO full and mode fault interrupt.*
- static void [SPI\\_HAL\\_SetTransmitIntCmd](#) (SPI\_Type \*base, bool enable)  
*Enables or disables the SPI transmit buffer/FIFO empty interrupt.*
- static void [SPI\\_HAL\\_SetMatchIntCmd](#) (SPI\_Type \*base, bool enable)  
*Enables or disables the SPI match interrupt.*

## Status

- static bool [SPI\\_HAL\\_GetIntStatusFlag](#) (SPI\_Type \*base, spi\_int\_status\_flag\_t flag)  
*Gets the SPI interrupt status flag state.*
- static bool [SPI\\_HAL\\_IsReadBuffFullPending](#) (SPI\_Type \*base)  
*Checks whether the read buffer/FIFO is full.*
- static bool [SPI\\_HAL\\_IsTxBuffEmptyPending](#) (SPI\_Type \*base)  
*Checks whether the transmit buffer/FIFO is empty.*
- static bool [SPI\\_HAL\\_IsModeFaultPending](#) (SPI\_Type \*base)  
*Checks whether a mode fault occurred.*
- void [SPI\\_HAL\\_ClearModeFaultFlag](#) (SPI\_Type \*base)  
*Clears the mode fault flag.*
- static bool [SPI\\_HAL\\_IsMatchPending](#) (SPI\_Type \*base)  
*Checks whether the data received matches the previously-set match value.*
- void [SPI\\_HAL\\_ClearMatchFlag](#) (SPI\_Type \*base)  
*Clears the match flag.*

## Data transfer

- static uint8\_t [SPI\\_HAL\\_ReadData](#) (SPI\_Type \*base)  
*Reads a byte from the data buffer.*
- static void [SPI\\_HAL\\_WriteData](#) (SPI\_Type \*base, uint8\_t data)  
*Writes a byte into the data buffer.*
- void [SPI\\_HAL\\_WriteDataBlocking](#) (SPI\_Type \*base, uint8\_t data)  
*Writes a byte into the data buffer and waits till complete to return.*

## Match byte

- static void [SPI\\_HAL\\_SetMatchValue](#) (SPI\_Type \*base, uint8\_t matchByte)  
*Sets the value which triggers the match interrupt.*

## SPI HAL driver

### 37.2.2 Enumeration Type Documentation

#### 37.2.2.1 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.

#### 37.2.2.2 enum spi\_master\_slave\_mode\_t

Enumerator

- kSpiMaster* SPI peripheral operates in master mode.
- kSpiSlave* SPI peripheral operates in slave mode.

#### 37.2.2.3 enum spi\_clock\_polarity\_t

Enumerator

- kSpiClockPolarity\_ActiveHigh* Active-high SPI clock (idles low).
- kSpiClockPolarity\_ActiveLow* Active-low SPI clock (idles high).

#### 37.2.2.4 enum spi\_clock\_phase\_t

Enumerator

- kSpiClockPhase\_FirstEdge* First edge on SPSCCK occurs at the middle of the first cycle of a data transfer.
- kSpiClockPhase\_SecondEdge* First edge on SPSCCK occurs at the start of the first cycle of a data transfer.

### 37.2.2.5 enum spi\_shift\_direction\_t

Enumerator

*kSpiMsbFirst* Data transfers start with most significant bit.

*kSpiLsbFirst* Data transfers start with least significant bit.

### 37.2.2.6 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.

### 37.2.2.7 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

### 37.2.2.8 enum spi\_data\_bitcount\_mode\_t

Enumerator

*kSpi8BitMode* 8-bit data transmission mode

*kSpi16BitMode* 16-bit data transmission mode

### 37.2.2.9 enum spi\_interrupt\_source\_t

Enumerator

*kSpiRxFullAndModfInt* Receive buffer full (SPRF) and mode fault (MODF) interrupt.

*kSpiTxEmptyInt* Transmit buffer empty interrupt.

*kSpiMatchInt* Match interrupt.

## SPI HAL driver

### 37.2.2.10 enum spi\_int\_status\_flag\_t

Enumerator

- kSpiRxBufferFullFlag* Read buffer full flag.
- kSpiMatchFlag* Match flag.
- kSpiTxBufferEmptyFlag* Transmit buffer empty flag.
- kSpiModeFaultFlag* Mode fault flag.

### 37.2.2.11 enum spi\_fifo\_interrupt\_source\_t

Enumerator

- kSpiRxFifoNearFullInt* Receive FIFO nearly full interrupt.
- kSpiTxFifoNearEmptyInt* Transmit FIFO nearly empty interrupt.

### 37.2.2.12 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.

### 37.2.2.13 enum spi\_txfifo\_watermark\_t

### 37.2.2.14 enum spi\_rxfifo\_watermark\_t

### 37.2.2.15 enum spi\_fifo\_status\_flag\_t

### 37.2.2.16 enum spi\_fifo\_error\_flag\_t

Enumerator

- kSpiNoFifoError* No error is detected.
- kSpiRxfof* Rx FIFO Overflow.
- 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.  
*kSpiRxfofTxfofRxferrTxferr* Rx FIFO Overflow, Tx FIFO Overflow Rx FIFO Error, Tx FIFO Error.

### 37.2.3 Function Documentation

#### 37.2.3.1 void SPI\_HAL\_Init ( SPI\_Type \* *base* )

This function basically resets all of the SPI registers to their default setting including disabling the module.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type. |
|-------------|---------------------------------------|

#### 37.2.3.2 static void SPI\_HAL\_Enable ( SPI\_Type \* *base* ) [inline], [static]

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type. |
|-------------|---------------------------------------|

#### 37.2.3.3 static void SPI\_HAL\_Disable ( SPI\_Type \* *base* ) [inline], [static]

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type. |
|-------------|---------------------------------------|

#### 37.2.3.4 uint32\_t SPI\_HAL\_SetBaud ( SPI\_Type \* *base*, 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).

## SPI HAL driver

### Parameters

|                         |                                           |
|-------------------------|-------------------------------------------|
| <i>base</i>             | Module base pointer of type SPI_Type.     |
| <i>bitsPerSec</i>       | The desired baud rate in bits per second. |
| <i>sourceClockIn-Hz</i> | Module source input clock in Hertz.       |

### Returns

The actual calculated baud rate in Hz.

### 37.2.3.5 static void SPI\_HAL\_SetBaudDivisors ( SPI\_Type \* *base*, 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>base</i>             | Module base pointer of type SPI_Type. |
| <i>prescale-Divisor</i> | baud rate prescale divisor setting.   |
| <i>rateDivisor</i>      | baud rate divisor setting.            |

### 37.2.3.6 static void SPI\_HAL\_SetMasterSlave ( SPI\_Type \* *base*, spi\_master\_slave\_mode\_t *mode* ) [inline], [static]

### Parameters

|             |                                                                   |
|-------------|-------------------------------------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type.                             |
| <i>mode</i> | Mode setting (master or slave) of type dsapi_master_slave_mode_t. |

### 37.2.3.7 static bool SPI\_HAL\_IsMaster ( SPI\_Type \* *base* ) [inline], [static]

## Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type. |
|-------------|---------------------------------------|

## Returns

true The module is in master mode. false The module is in slave mode.

**37.2.3.8 void SPI\_HAL\_SetSlaveSelectOutputMode ( SPI\_Type \* *base*, 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>base</i> | Module base pointer of type SPI_Type.                               |
| <i>mode</i> | Selection input of one of three modes of type spi_ss_output_mode_t. |

**37.2.3.9 void SPI\_HAL\_SetDataFormat ( SPI\_Type \* *base*, 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>base</i>      | Module base pointer of type SPI_Type.                            |
| <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. |

**37.2.3.10 static uint32\_t SPI\_HAL\_GetDataRegAddr ( SPI\_Type \* *base* ) [inline], [static]**

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

## SPI HAL driver

### Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type. |
|-------------|---------------------------------------|

### Returns

The SPI data register address.

### 37.2.3.11 void SPI\_HAL\_SetPinMode ( SPI\_Type \* *base*, 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>base</i> | Module base pointer of type SPI_Type.           |
| <i>mode</i> | Operational of SPI pins of type spi_pin_mode_t. |

### 37.2.3.12 void SPI\_HAL\_SetIntMode ( SPI\_Type \* *base*, spi\_interrupt\_source\_t *interrupt*, bool *enable* )

This function enables or disables the SPI receive buffer (or FIFO if the module supports a FIFO) full and mode fault interrupt SPI transmit buffer (or FIFO if the module supports a FIFO) empty interrupt SPI match interrupt

Example, to set the receive and mode fault interrupt: SPI\_HAL\_SetIntMode(base, kSpiRxFullAndModf-Int, true);

### Parameters

|                  |                                                                                    |
|------------------|------------------------------------------------------------------------------------|
| <i>base</i>      | Module base pointer of type SPI_Type.                                              |
| <i>interrupt</i> | SPI interrupt source to configure of type spi_interrupt_source_t.                  |
| <i>enable</i>    | Enable (true) or disable (false) the receive buffer full and mode fault interrupt. |

### 37.2.3.13 static void SPI\_HAL\_SetReceiveAndFaultIntCmd ( SPI\_Type \* *base*, 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>base</i>   | Module base pointer of type SPI_Type.                                              |
| <i>enable</i> | Enable (true) or disable (false) the receive buffer full and mode fault interrupt. |

#### 37.2.3.14 **static void SPI\_HAL\_SetTransmitIntCmd ( SPI\_Type \* *base*, 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>base</i>   | Module base pointer of type SPI_Type.                                 |
| <i>enable</i> | Enable (true) or disable (false) the transmit buffer empty interrupt. |

#### 37.2.3.15 **static void SPI\_HAL\_SetMatchIntCmd ( SPI\_Type \* *base*, bool *enable* ) [inline], [static]**

This function enables or disables the SPI match interrupt.

Parameters

|               |                                                       |
|---------------|-------------------------------------------------------|
| <i>base</i>   | Module base pointer of type SPI_Type.                 |
| <i>enable</i> | Enable (true) or disable (false) the match interrupt. |

#### 37.2.3.16 **static bool SPI\_HAL\_GetIntStatusFlag ( SPI\_Type \* *base*, spi\_int\_status\_flag\_t *flag* ) [inline], [static]**

This function returns the state (set or cleared) of the SPI interrupt status flag.

Parameters

|             |                                                                    |
|-------------|--------------------------------------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type.                              |
| <i>flag</i> | The requested interrupt status flag of type spi_int_status_flag_t. |

Returns

Current setting of the requested interrupt status flag.

## SPI HAL driver

### 37.2.3.17 static bool SPI\_HAL\_IsReadBuffFullPending ( SPI\_Type \* *base* ) [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(base))
{
 // Read the data in the buffer, which also clears the flag.
 byte = SPI_HAL_ReadData(base);
}
```

#### Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type. |
|-------------|---------------------------------------|

#### Returns

Current setting of the read buffer full flag.

### 37.2.3.18 static bool SPI\_HAL\_IsTxBuffEmptyPending ( SPI\_Type \* *base* ) [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(base))
{
 // Buffer has room, so write the next data value.
 SPI_HAL_WriteData(base, byte);
}
```

#### Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type. |
|-------------|---------------------------------------|

#### Returns

Current setting of the transmit buffer empty flag.

### 37.2.3.19 static bool SPI\_HAL\_IsModeFaultPending ( SPI\_Type \* *base* ) [inline], [static]

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type. |
|-------------|---------------------------------------|

Returns

Current setting of the mode fault flag.

**37.2.3.20 void SPI\_HAL\_ClearModeFaultFlag ( SPI\_Type \* *base* )**

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type |
|-------------|--------------------------------------|

**37.2.3.21 static bool SPI\_HAL\_IsMatchPending ( SPI\_Type \* *base* ) [inline], [static]**

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type. |
|-------------|---------------------------------------|

Returns

Current setting of the match flag.

**37.2.3.22 void SPI\_HAL\_ClearMatchFlag ( SPI\_Type \* *base* )**

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type. |
|-------------|---------------------------------------|

**37.2.3.23 static uint8\_t SPI\_HAL\_ReadData ( SPI\_Type \* *base* ) [inline], [static]**

Parameters

---

## SPI HAL driver

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type. |
|-------------|---------------------------------------|

Returns

The data read from the data buffer.

**37.2.3.24 static void SPI\_HAL\_WriteData ( SPI\_Type \* *base*, uint8\_t *data* ) [inline], [static]**

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type. |
| <i>data</i> | The data to send.                     |

**37.2.3.25 void SPI\_HAL\_WriteDataBlocking ( SPI\_Type \* *base*, uint8\_t *data* )**

This function writes data to the SPI data register and waits until the TX is empty to return.

Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Module base pointer of type SPI_Type. |
| <i>data</i> | The data to send.                     |

**37.2.3.26 static void SPI\_HAL\_SetMatchValue ( SPI\_Type \* *base*, uint8\_t *matchByte* ) [inline], [static]**

Parameters

|                  |                                               |
|------------------|-----------------------------------------------|
| <i>base</i>      | Module base pointer of type SPI_Type.         |
| <i>matchByte</i> | The value which triggers the match interrupt. |



## 37.3 SPI Master Peripheral Driver

### 37.3.1 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, subsequent chapters refer to either set of drivers as the "SPI master driver" when discussing items that pertain to either driver. Note, when using the DMA-driven SPI driver, 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 because the SPI interrupt handler needs to be changed. 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

## SPI Master Peripheral Driver

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.

### 37.3.2 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.

### 37.3.3 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).

### 37.3.4 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) and both examples are provided below. First, the interrupt-driven example is 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
```

## SPI Master Peripheral Driver

### 37.3.5 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);
```

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);
```

### 37.3.6 SPI De-initialization

To de-initialize and shut down the SPI module, call the function:

```
// interrupt-driven
```

## SPI Master Peripheral Driver

```
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

- SPI\_Type \*const [g\\_spiBase](#) [SPI\_INSTANCE\_COUNT]  
*Table of base pointers for SPI instances.*
- const IRQn\_Type [g\\_spiIrqId](#) [SPI\_INSTANCE\_COUNT]  
*Table to save SPI IRQ enumeration numbers defined in CMSIS header file.*
- SPI\_Type \*const [g\\_spiBase](#) [SPI\_INSTANCE\_COUNT]  
*Table of base pointers for SPI instances.*
- const IRQn\_Type [g\\_spiIrqId](#) [SPI\_INSTANCE\_COUNT]  
*Table to save SPI IRQ enumeration numbers defined in the CMSIS header file.*

### Initialization and shutdown

- [spi\\_status\\_t SPI\\_DRV\\_DmaMasterInit](#) (uint32\_t instance, [spi\\_dma\\_master\\_state\\_t](#) \*spiDmaState)  
*Initializes a SPI instance for master mode operation to work with DMA.*
- [spi\\_status\\_t 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.*

## Blocking transfers

- `spi_status_t SPI_DRV_DmaMasterTransferBlocking` (uint32\_t instance, const `spi_dma_master_user_config_t` \*device, const uint8\_t \*sendBuffer, uint8\_t \*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` \*device, const uint8\_t \*sendBuffer, uint8\_t \*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 \*bytes-Transferred)  
*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.*
- void `SPI_DRV_DmaMasterIRQHandler` (uint32\_t instance)  
*Interrupt handler for SPI master mode.*

## Initialization and shutdown

- `spi_status_t SPI_DRV_MasterInit` (uint32\_t instance, `spi_master_state_t` \*spiState)  
*Initializes an SPI instance for master mode operation.*
- `spi_status_t 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` \*device, const uint8\_t \*sendBuffer, uint8\_t \*receiveBuffer, size\_t transferByteCount, uint32\_t timeout)  
*Performs a blocking SPI master mode transfer.*

## SPI Master Peripheral Driver

### Non-blocking transfers

- [spi\\_status\\_t SPI\\_DRV\\_MasterTransfer](#) (uint32\_t instance, const [spi\\_master\\_user\\_config\\_t](#) \*device, const uint8\_t \*sendBuffer, uint8\_t \*receiveBuffer, size\_t transferByteCount)  
*Performs a non-blocking SPI master mode transfer.*
- [spi\\_status\\_t SPI\\_DRV\\_MasterGetTransferStatus](#) (uint32\_t instance, uint32\_t \*bytesTransferred)  
*Returns whether the previous transfer is completed.*
- [spi\\_status\\_t SPI\\_DRV\\_MasterAbortTransfer](#) (uint32\_t instance)  
*Terminates an asynchronous transfer early.*
- void [SPI\\_DRV\\_MasterIRQHandler](#) (uint32\_t instance)  
*Interrupt handler for SPI master mode.*

### 37.3.7 Data Structure Documentation

#### 37.3.7.1 struct spi\_dma\_master\_user\_config\_t

##### Data Fields

- uint32\_t [bitsPerSec](#)  
*SPI baud rate in bits per sec.*

#### 37.3.7.2 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 \* [sendBuffer](#)  
*The buffer being sent.*
- uint8\_t \* [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.*
- `uint32_t transferByteCnt`  
*Number of bytes to transfer.*

### 37.3.7.2.0.61 Field Documentation

37.3.7.2.0.61.1 `volatile bool spi_dma_master_state_t::isTransferInProgress`

37.3.7.2.0.61.2 `const uint8_t* spi_dma_master_state_t::sendBuffer`

37.3.7.2.0.61.3 `uint8_t* spi_dma_master_state_t::receiveBuffer`

37.3.7.2.0.61.4 `volatile size_t spi_dma_master_state_t::remainingSendByteCount`

37.3.7.2.0.61.5 `volatile size_t spi_dma_master_state_t::remainingReceiveByteCount`

37.3.7.2.0.61.6 `volatile size_t spi_dma_master_state_t::transferredByteCount`

37.3.7.2.0.61.7 `volatile bool spi_dma_master_state_t::isTransferBlocking`

37.3.7.2.0.61.8 `semaphore_t spi_dma_master_state_t::irqSync`

37.3.7.2.0.61.9 `uint32_t spi_dma_master_state_t::transferByteCnt`

### 37.3.7.3 struct `spi_master_user_config_t`

#### Data Fields

- `uint32_t bitsPerSec`  
*SPI baud rate in bits per sec.*
- `spi_clock_polarity_t polarity`  
*Active high or low clock polarity.*
- `spi_clock_phase_t phase`  
*Clock phase setting to change and capture data.*
- `spi_shift_direction_t direction`  
*MSB or LSB data shift direction.*

### 37.3.7.4 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.

## SPI Master Peripheral Driver

### Data Fields

- uint32\_t `spiSourceClock`  
*Module source clock.*
- volatile bool `isTransferInProgress`  
*True if there is an active transfer.*
- const uint8\_t \* `sendBuffer`  
*The buffer being sent.*
- uint8\_t \* `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.*

#### 37.3.7.4.0.62 Field Documentation

- 37.3.7.4.0.62.1 volatile bool `spi_master_state_t::isTransferInProgress`
- 37.3.7.4.0.62.2 const uint8\_t\* `spi_master_state_t::sendBuffer`
- 37.3.7.4.0.62.3 uint8\_t\* `spi_master_state_t::receiveBuffer`
- 37.3.7.4.0.62.4 volatile size\_t `spi_master_state_t::remainingSendByteCount`
- 37.3.7.4.0.62.5 volatile size\_t `spi_master_state_t::remainingReceiveByteCount`
- 37.3.7.4.0.62.6 volatile size\_t `spi_master_state_t::transferredByteCount`
- 37.3.7.4.0.62.7 volatile bool `spi_master_state_t::isTransferBlocking`
- 37.3.7.4.0.62.8 semaphore\_t `spi_master_state_t::irqSync`

### 37.3.8 Enumeration Type Documentation

#### 37.3.8.1 enum `_spi_dma_timeouts`

Enumerator

`kSpiDmaWaitForever` Waits forever for a transfer to complete.

### 37.3.8.2 enum \_spi\_timeouts

Enumerator

*kSpiWaitForever* Waits forever for a transfer to complete.

### 37.3.9 Function Documentation

#### 37.3.9.1 spi\_status\_t 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, un-gates 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. |

Returns

kStatus\_SPI\_Success indicating successful initialization

#### 37.3.9.2 spi\_status\_t 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. |
|-----------------|--------------------------------------------|

Returns

kStatus\_SPI\_Success indicating successful de-initialization

## SPI Master Peripheral Driver

### 37.3.9.3 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. |

### 37.3.9.4 spi\_status\_t SPI\_DRV\_DmaMasterTransferBlocking ( uint32\_t *instance*, const spi\_dma\_master\_user\_config\_t \* *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 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. |

### 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.

### 37.3.9.5 `spi_status_t SPI_DRV_DmaMasterTransfer ( uint32_t instance, const spi_dma_master_user_config_t * device, const uint8_t * sendBuffer, uint8_t * 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 not 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.

## SPI Master Peripheral Driver

### 37.3.9.6 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.

### 37.3.9.7 spi\_status\_t SPI\_DRV\_DmaMasterAbortTransfer ( 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 SPI peripheral. |
|-----------------|--------------------------------------------|

#### Returns

kStatus\_SPI\_Success The transfer was successful. kStatus\_SPI\_NoTransferInProgress No transfer is currently in progress.

### 37.3.9.8 void SPI\_DRV\_DmaMasterIRQHandler ( uint32\_t instance )

This handler is used when the extraByte flag is set to retrieve the received last byte.

#### Parameters

---

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the SPI peripheral. |
|-----------------|--------------------------------------------|

### 37.3.9.9 **spi\_status\_t 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, un-gates 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. |

Returns

kStatus\_SPI\_Success indicating successful initialization

### 37.3.9.10 **spi\_status\_t SPI\_DRV\_MasterDeinit ( uint32\_t *instance* )**

This function resets the SPI peripheral, gates its clock, and disables the interrupt to the core.

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the SPI peripheral. |
|-----------------|--------------------------------------------|

Returns

kStatus\_SPI\_Success indicating successful de-initialization

### 37.3.9.11 **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

---

## SPI Master Peripheral Driver

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 is returned. |

### 37.3.9.12 spi\_status\_t SPI\_DRV\_MasterTransferBlocking ( uint32\_t instance, const spi\_master\_user\_config\_t \* 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 a full-duplex bus, and does not 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.

### 37.3.9.13 `spi_status_t SPI_DRV_MasterTransfer ( uint32_t instance, const spi_master_user_config_t * device, const uint8_t * sendBuffer, uint8_t * 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 not 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.

### 37.3.9.14 `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 |

### 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.

#### 37.3.9.15 spi\_status\_t SPI\_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 SPI peripheral. |
|-----------------|--------------------------------------------|

### Returns

kStatus\_SPI\_Success The transfer was successful. kStatus\_SPI\_NoTransferInProgress No transfer is currently in progress.

#### 37.3.9.16 void SPI\_DRV\_MasterIRQHandler ( uint32\_t instance )

This handler uses the buffers stored in the [spi\\_master\\_state\\_t](#) structs to transfer data.

### Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the SPI peripheral. |
|-----------------|--------------------------------------------|

### 37.3.10 Variable Documentation

**37.3.10.1** SPI\_Type\* const g\_spiBase[SPI\_INSTANCE\_COUNT]

**37.3.10.2** const IRQn\_Type g\_spilrqld[SPI\_INSTANCE\_COUNT]

**37.3.10.3** SPI\_Type\* const g\_spiBase[SPI\_INSTANCE\_COUNT]

**37.3.10.4** const IRQn\_Type g\_spilrqld[SPI\_INSTANCE\_COUNT]

## SPI Slave Peripheral Driver

### 37.4 SPI Slave Peripheral Driver

#### 37.4.1 Overview

This section describes the programming interface of the SPI slave mode peripheral driver.

#### 37.4.2 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, subsequent chapters 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, 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 because the SPI interrupt handler needs to be changed. 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)`.

### 37.4.3 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.

### 37.4.4 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.

### 37.4.5 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
#if FSL_FEATURE_SPI_16BIT_TRANSFERS
 slaveDmaUserConfig.bitCount = kSpi8BitMode;
#endif

 // init the slave (DMA-driven)
 SPI_DRV_DmaSlaveInit(instance, &spiDmaSlaveState, &slaveDmaUserConfig);
#endif

```

### 37.4.6 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 success 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 success 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.

```

### 37.4.7 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 sends when transmit data was not configured.*
- #define [SPI\\_DEFAULT\\_DUMMY\\_PATTERN](#) (0x0U)  
*Dummy pattern, that SPI slave sends when transmit data was not configured.*

### Variables

- SPI\_Type \*const [g\\_spiBase](#) [SPI\_INSTANCE\_COUNT]

## SPI Slave Peripheral Driver

- *Table of base pointers for SPI instances.*  
const IRQn\_Type [g\\_spiIrqId](#) [SPI\_INSTANCE\_COUNT]
- *Table to save SPI IRQ enumeration numbers defined in CMSIS header file.*  
SPI\_Type \*const [g\\_spiBase](#) [SPI\_INSTANCE\_COUNT]
- *Table of base pointers for SPI instances.*  
const IRQn\_Type [g\\_spiIrqId](#) [SPI\_INSTANCE\_COUNT]
- *Table to save SPI IRQ enumeration numbers defined in the CMSIS header file.*

### 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.*
- [spi\\_status\\_t 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 the SPI bus using an interrupt and a non-blocking call.*
- [spi\\_status\\_t SPI\\_DRV\\_DmaSlaveAbortTransfer](#) (uint32\_t instance)  
*Aborts the transfer that started by a non-blocking call transfer function.*
- [spi\\_status\\_t SPI\\_DRV\\_DmaSlaveGetTransferStatus](#) (uint32\_t instance, uint32\_t \*frames-Transferred)  
*Returns whether the previous transfer is finished.*
- void [SPI\\_DRV\\_DmaSlaveIRQHandler](#) (uint32\_t instance)  
*Interrupt handler for SPI slave mode.*

### 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.*
- [spi\\_status\\_t SPI\\_DRV\\_SlaveDeinit](#) (uint32\_t instance)  
*Shuts down an 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 a 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 the transfer data on SPI bus using an interrupt and a non-blocking call.*
- [spi\\_status\\_t SPI\\_DRV\\_SlaveAbortTransfer](#) (uint32\_t instance)  
*Aborts the transfer that started by a non-blocking call transfer function.*
- [spi\\_status\\_t SPI\\_DRV\\_SlaveGetTransferStatus](#) (uint32\_t instance, uint32\_t \*framesTransferred)  
*Returns whether the previous transfer is finished.*
- void [SPI\\_DRV\\_SlaveIRQHandler](#) (uint32\_t instance)  
*SPI Slave Generic IRQ handler.*

## 37.4.8 Data Structure Documentation

### 37.4.8.1 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.*

#### 37.4.8.1.0.63 Field Documentation

37.4.8.1.0.63.1 [spi\\_clock\\_phase\\_t spi\\_dma\\_slave\\_user\\_config\\_t::phase](#)

37.4.8.1.0.63.2 [spi\\_clock\\_polarity\\_t spi\\_dma\\_slave\\_user\\_config\\_t::polarity](#)

37.4.8.1.0.63.3 [spi\\_shift\\_direction\\_t spi\\_dma\\_slave\\_user\\_config\\_t::direction](#)

### 37.4.8.2 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.

## SPI Slave Peripheral Driver

### 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 is sent when there is no data in the transmit buffer.*
- [volatile bool isTransferInProgress](#)  
*True if there is an active transfer.*
- [const uint8\\_t \\* sendBuffer](#)  
*Pointer to transmit buffer.*
- [uint8\\_t \\* 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 a-sync.*
- [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.*

### 37.4.8.2.0.64 Field Documentation

**37.4.8.2.0.64.1** [volatile bool spi\\_dma\\_slave\\_state\\_t::isTransferInProgress](#)

**37.4.8.2.0.64.2** [volatile int32\\_t spi\\_dma\\_slave\\_state\\_t::remainingSendByteCount](#)

**37.4.8.2.0.64.3** [volatile int32\\_t spi\\_dma\\_slave\\_state\\_t::remainingReceiveByteCount](#)

**37.4.8.2.0.64.4** [volatile int32\\_t spi\\_dma\\_slave\\_state\\_t::transferredByteCount](#)

### 37.4.8.3 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.*

#### 37.4.8.4 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 is sent when there is no data in the transmit buffer.*
- `volatile bool isTransferInProgress`  
*True if there is an active transfer.*
- `const uint8_t * sendBuffer`  
*Pointer to transmit buffer.*
- `uint8_t * 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 a-sync.*

## SPI Slave Peripheral Driver

### 37.4.8.4.0.65 Field Documentation

37.4.8.4.0.65.1 volatile bool `spi_slave_state_t::isTransferInProgress`

37.4.8.4.0.65.2 volatile int32\_t `spi_slave_state_t::remainingSendByteCount`

37.4.8.4.0.65.3 volatile int32\_t `spi_slave_state_t::remainingReceiveByteCount`

37.4.8.4.0.65.4 volatile int32\_t `spi_slave_state_t::transferredByteCount`

### 37.4.9 Function Documentation

**37.4.9.1** `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 uses an 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`.

**37.4.9.2** `spi_status_t SPI_DRV_DmaSlaveDeinit ( uint32_t instance )`

Disables the 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. |
|-----------------|--------------------------------------------|

Returns

`kStatus_SPI_Success` indicating successful de-initialization

### 37.4.9.3 `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 is ignored. If the `receiveBuffer` is NULL, the receive process is ignored. If both the `receiveBuffer` and the `sendBuffer` are available, the transmit and the receive progress are processed. If only the `receiveBuffer` available, the receive is processed. Otherwise, the transmit is processed. This function returns when its processes are 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 milliseconds that function waits 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.

### 37.4.9.4 `spi_status_t SPI_DRV_DmaSlaveTransfer ( uint32_t instance, const uint8_t * sendBuffer, uint8_t * receiveBuffer, uint32_t transferByteCount )`

This function checks the driver status then set buffer pointers to receive and transmit SPI data. If the `sendBuffer` is NULL, the transmit process is ignored. If the `receiveBuffer` is NULL, the receive process is ignored. If both the `receiveBuffer` and the `sendBuffer` available, transfer is done when the `kDspiTxDone` and `kDspiRxDone` are set. If only the `receiveBuffer` is available, the transfer is done when the `kDspiRxDone` flag is set. Otherwise, the transfer is done when the `kDspiTxDone` was set. This function uses an 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.

### 37.4.9.5 spi\_status\_t SPI\_DRV\_DmaSlaveAbortTransfer ( uint32\_t instance )

This function stops the transfer which was started by the [SPI\\_DRV\\_SlaveTransfer\(\)](#) function.

#### 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.

### 37.4.9.6 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.

**37.4.9.7 void SPI\_DRV\_DmaSlaveIRQHandler ( uint32\_t *instance* )**

This handler is used when the hasExtraByte flag is set to retrieve the received last byte.

## SPI Slave Peripheral Driver

### Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the SPI peripheral. |
|-----------------|--------------------------------------------|

### 37.4.9.8 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. After it is initialized, the SPI module is configured in slave mode and the user can start transmitting and receiving data by calling send, receive, and transfer functions. This function indicates SPI slave uses an 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.

### 37.4.9.9 spi\_status\_t SPI\_DRV\_SlaveDeinit ( uint32\_t instance )

Disables the SPI module, gates its clock, and changes the SPI slave driver state to NonInit for the SPI slave module which is initialized with interrupt mechanism. After de-initialization, the user can re-initialize the SPI slave module with other mechanisms.

### Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>instance</i> | The instance number of the SPI peripheral. |
|-----------------|--------------------------------------------|

### Returns

An error code or kStatus\_SPI\_Success.

### 37.4.9.10 spi\_status\_t SPI\_DRV\_SlaveTransferBlocking ( uint32\_t instance, const uint8\_t \* sendBuffer, uint8\_t \* receiveBuffer, uint32\_t transferByteCount, uint32\_t timeout )

This function checks the driver status and mechanism, and transmits/receives data through the SPI bus. If the sendBuffer is NULL, the transmit process is ignored. If the receiveBuffer is NULL, the receive process



is ignored. If both the receiveBuffer and the sendBuffer are available, the transmit and the receive progress is processed. If only the receiveBuffer is available, the receive is processed. Otherwise, the transmit is processed. This function only returns when the processes are completed. This function uses an 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 milliseconds that function waits 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.

### 37.4.9.11 spi\_status\_t SPI\_DRV\_SlaveTransfer ( uint32\_t instance, const uint8\_t \* sendBuffer, uint8\_t \* receiveBuffer, uint32\_t transferByteCount )

This function checks the driver status and sets buffer pointers to receive and transmit SPI data. If the sendBuffer is NULL, the transmit process is ignored. If the receiveBuffer is NULL, the receive process is ignored. If both the receiveBuffer and the sendBuffer are available, the transfer is done when the kDspi-TxDone and kDspiRxDone are set. If only the receiveBuffer is available, the transfer is done when the kDspiRxDone flag is set. Otherwise, the transfer is done when the kDspiTxDone was set. This function uses an 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.                    |

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.

## SPI Slave Peripheral Driver

### 37.4.9.12 spi\_status\_t SPI\_DRV\_SlaveAbortTransfer ( uint32\_t *instance* )

This function stops the transfer which was started by the calling the [SPI\\_DRV\\_SlaveTransfer\(\)](#) function.

### 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.

#### 37.4.9.13 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.

#### 37.4.9.14 void SPI\_DRV\_SlaveIRQHandler ( uint32\_t *instance* )

### Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>instance</i> | Instance number of the SPI module. |
|-----------------|------------------------------------|

## SPI Slave Peripheral Driver

### 37.4.10 Variable Documentation

37.4.10.1 SPI\_Type\* const g\_spiBase[SPI\_INSTANCE\_COUNT]

37.4.10.2 const IRQn\_Type g\_spilrqld[SPI\_INSTANCE\_COUNT]

37.4.10.3 SPI\_Type\* const g\_spiBase[SPI\_INSTANCE\_COUNT]

37.4.10.4 const IRQn\_Type g\_spilrqld[SPI\_INSTANCE\_COUNT]

## **37.5 Shared SPI Types**

This chapter describes SPI driver shared types.

## SPI Classes

### 37.6 SPI Classes

This chapter describes the SPI driver C++ classes.

## Chapter 38

### Timer/PWM Module (TPM)

#### 38.1 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

### 38.2 TPM HAL driver

#### 38.2.1 Overview

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

#### Data Structures

- struct `tpm_pwm_param_t`  
*TPM 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` {  
`kTpmTrigSel0 = 0,`  
`kTpmTrigSel1,`  
`kTpmTrigSel2,`  
`kTpmTrigSel3,`  
`kTpmTrigSel4,`  
`kTpmTrigSel5,`  
`kTpmTrigSel6,`  
`kTpmTrigSel7,`  
`kTpmTrigSel8,`  
`kTpmTrigSel9,`  
`kTpmTrigSel10,`  
`kTpmTrigSel11,`  
`kTpmTrigSel12,`  
`kTpmTrigSel13,`  
`kTpmTrigSel14,`  
`kTpmTrigSel15 }`  
*TPM trigger sources, please refer to the chip reference manual for available options.*
- enum `tpm_pwm_mode_t` {  
`kTpmEdgeAlignedPWM = 0,`  
`kTpmCenterAlignedPWM }`  
*TPM operation mode.*
- enum `tpm_pwm_edge_mode_t` {  
`kTpmHighTrue = 0,`  
`kTpmLowTrue }`  
*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.*
- enum `tpm_status_t` {  
`kStatusTpmSuccess = 0x00U`,  
`kStatusTpmFail = 0x01U` }  
*Error codes for TPM driver.*

## Functions

- void `TPM_HAL_Reset` (TPM\_Type \*tpmBase, uint32\_t instance)  
*reset tpm registers*
- void `TPM_HAL_EnablePwmMode` (TPM\_Type \*tpmBase, `tpm_pwm_param_t` \*config, uint8\_t channel)  
*Enables the TPM PWM output mode.*
- void `TPM_HAL_DisableChn` (TPM\_Type \*tpmBase, uint8\_t channel)  
*Disables the TPM channel.*
- void `TPM_HAL_SetClockMode` (TPM\_Type \*tpmBase, `tpm_clock_mode_t` mode)  
*Set TPM clock mode.*
- static `tpm_clock_mode_t` `TPM_HAL_GetClockMode` (TPM\_Type \*tpmBase)  
*get TPM clock mode.*
- static void `TPM_HAL_SetClockDiv` (TPM\_Type \*tpmBase, `tpm_clock_ps_t` ps)  
*set TPM clock divider.*
- static `tpm_clock_ps_t` `TPM_HAL_GetClockDiv` (TPM\_Type \*tpmBase)  
*get TPM clock divider.*
- static void `TPM_HAL_EnableTimerOverflowInt` (TPM\_Type \*tpmBase)  
*Enable the TPM peripheral timer overflow interrupt.*
- static void `TPM_HAL_DisableTimerOverflowInt` (TPM\_Type \*tpmBase)  
*Disable the TPM peripheral timer overflow interrupt.*
- static bool `TPM_HAL_IsOverflowIntEnabled` (TPM\_Type \*tpmBase)  
*Read the bit that controls TPM timer overflow interrupt enablement.*
- static bool `TPM_HAL_GetTimerOverflowStatus` (TPM\_Type \*tpmBase)  
*return TPM peripheral timer overflow interrupt flag.*
- static void `TPM_HAL_ClearTimerOverflowFlag` (TPM\_Type \*tpmBase)  
*Clear the TPM timer overflow interrupt flag.*
- static void `TPM_HAL_SetCpwms` (TPM\_Type \*tpmBase, uint8\_t mode)  
*set TPM center-aligned PWM select.*
- static bool `TPM_HAL_GetCpwms` (TPM\_Type \*tpmBase)  
*get TPM center-aligned PWM selection value.*
- static void `TPM_HAL_ClearCounter` (TPM\_Type \*tpmBase)  
*clear TPM peripheral current counter value.*
- static uint16\_t `TPM_HAL_GetCounterVal` (TPM\_Type \*tpmBase)  
*return TPM peripheral current counter value.*
- static void `TPM_HAL_SetMod` (TPM\_Type \*tpmBase, uint16\_t val)  
*set TPM peripheral timer modulo value.*
- static uint16\_t `TPM_HAL_GetMod` (TPM\_Type \*tpmBase)  
*return TPM peripheral counter modulo value.*
- static void `TPM_HAL_SetChnMsnbaElsnbaVal` (TPM\_Type \*tpmBase, uint8\_t channel, uint8\_t

## TPM HAL driver

value)

*Set TPM peripheral timer channel mode and edge level.*

- static uint8\_t [TPM\\_HAL\\_GetChnMsnbaVal](#) (TPM\_Type \*tpmBase, uint8\_t channel)  
*get TPM peripheral timer channel mode.*
- static uint8\_t [TPM\\_HAL\\_GetChnElsnbaVal](#) (TPM\_Type \*tpmBase, uint8\_t channel)  
*get TPM peripheral timer channel edge level.*
- static void [TPM\\_HAL\\_EnableChnInt](#) (TPM\_Type \*tpmBase, uint8\_t channel)  
*enable TPM peripheral timer channel(n) interrupt.*
- static void [TPM\\_HAL\\_DisableChnInt](#) (TPM\_Type \*tpmBase, uint8\_t channel)  
*disable TPM peripheral timer channel(n) interrupt.*
- static bool [TPM\\_HAL\\_IsChnIntEnabled](#) (TPM\_Type \*tpmBase, uint8\_t channel)  
*get TPM peripheral timer channel(n) interrupt enabled or not.*
- static bool [TPM\\_HAL\\_GetChnStatus](#) (TPM\_Type \*tpmBase, uint8\_t channel)  
*return if any event for TPM peripheral timer channel has occurred ,*
- static void [TPM\\_HAL\\_ClearChnInt](#) (TPM\_Type \*tpmBase, uint8\_t channel)  
*return if any event for TPM peripheral timer channel has occurred ,*
- static void [TPM\\_HAL\\_SetChnCountVal](#) (TPM\_Type \*tpmBase, uint8\_t channel, uint16\_t val)  
*set TPM peripheral timer channel counter value,*
- static uint16\_t [TPM\\_HAL\\_GetChnCountVal](#) (TPM\_Type \*tpmBase, uint8\_t channel)  
*get TPM peripheral timer channel counter value.*
- static uint32\_t [TPM\\_HAL\\_GetStatusRegVal](#) (TPM\_Type \*tpmBase)  
*get TPM peripheral timer channel event status.*
- static void [TPM\\_HAL\\_ClearStatusReg](#) (TPM\_Type \*tpmBase, uint16\_t tpm\_status)  
*clear TPM peripheral timer clear status register value,*
- static void [TPM\\_HAL\\_SetTriggerSrc](#) (TPM\_Type \*tpmBase, [tpm\\_trigger\\_source\\_t](#) trigger\_num)  
*set TPM peripheral timer trigger.*
- static void [TPM\\_HAL\\_SetTriggerMode](#) (TPM\_Type \*tpmBase, bool enable)  
*set TPM peripheral timer running on trigger or not .*
- static void [TPM\\_HAL\\_SetReloadOnTriggerMode](#) (TPM\_Type \*tpmBase, bool enable)  
*enable TPM timer counter reload on selected trigger or not.*
- static void [TPM\\_HAL\\_SetStopOnOverflowMode](#) (TPM\_Type \*tpmBase, bool enable)  
*enable TPM timer counter stop on selected trigger or not.*
- static void [TPM\\_HAL\\_EnableGlobalTimeBase](#) (TPM\_Type \*tpmBase, bool enable)  
*enable TPM timer global time base.*
- static void [TPM\\_HAL\\_SetDbgMode](#) (TPM\_Type \*tpmBase, bool enable)  
*set BDM mode.*
- static void [TPM\\_HAL\\_SetWaitMode](#) (TPM\_Type \*tpmBase, bool enable)  
*set WAIT mode behavior.*

## Variables

- const uint32\_t [g\\_tpmChannelCount](#) [TPM\_INSTANCE\_COUNT]  
*Table of number of channels for each TPM instance.*

## 38.2.2 Data Structure Documentation

### 38.2.2.1 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)...*

#### 38.2.2.1.0.66 Field Documentation

##### 38.2.2.1.0.66.1 uint32\_t tpm\_pwm\_param\_t::uDutyCyclePercent

100=active signal (100% duty cycle).

## 38.2.3 Enumeration Type Documentation

### 38.2.3.1 enum tpm\_clock\_mode\_t

### 38.2.3.2 enum tpm\_trigger\_source\_t

Enumerator

- kTpmTrigSel0* TPM trigger source 0.
- kTpmTrigSel1* TPM trigger source 1.
- kTpmTrigSel2* TPM trigger source 2.
- kTpmTrigSel3* TPM trigger source 3.
- kTpmTrigSel4* TPM trigger source 4.
- kTpmTrigSel5* TPM trigger source 5.
- kTpmTrigSel6* TPM trigger source 6.
- kTpmTrigSel7* TPM trigger source 7.
- kTpmTrigSel8* TPM trigger source 8.
- kTpmTrigSel9* TPM trigger source 8.
- kTpmTrigSel10* TPM trigger source 10.
- kTpmTrigSel11* TPM trigger source 11.
- kTpmTrigSel12* TPM trigger source 12.
- kTpmTrigSel13* TPM trigger source 13.
- kTpmTrigSel14* TPM trigger source 14.
- kTpmTrigSel15* TPM trigger source 15.

## TPM HAL driver

### 38.2.3.3 enum tpm\_pwm\_mode\_t

Enumerator

*kTpmEdgeAlignedPWM* Edge aligned mode.  
*kTpmCenterAlignedPWM* Center aligned mode.

### 38.2.3.4 enum tpm\_pwm\_edge\_mode\_t

Enumerator

*kTpmHighTrue* Clear output on match, set output on reload.  
*kTpmLowTrue* Set output on match, clear output on reload.

### 38.2.3.5 enum tpm\_status\_t

Enumerator

*kStatusTpmSuccess* TPM success status.  
*kStatusTpmFail* TPM error status.

## 38.2.4 Function Documentation

### 38.2.4.1 void TPM\_HAL\_Reset ( TPM\_Type \* *tpmBase*, uint32\_t *instance* )

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>tpmBase</i>  | TPM module base address pointer     |
| <i>instance</i> | The TPM peripheral instance number. |

### 38.2.4.2 void TPM\_HAL\_EnablePwmMode ( TPM\_Type \* *tpmBase*, tpm\_pwm\_param\_t \* *config*, uint8\_t *channel* )

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
|----------------|---------------------------------|

|                |                             |
|----------------|-----------------------------|
| <i>config</i>  | PWM configuration parameter |
| <i>channel</i> | The TPM channel number.     |

### 38.2.4.3 void TPM\_HAL\_DisableChn ( TPM\_Type \* *tpmBase*, uint8\_t *channel* )

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
| <i>channel</i> | The TPM channel number.         |

### 38.2.4.4 void TPM\_HAL\_SetClockMode ( TPM\_Type \* *tpmBase*, tpm\_clock\_mode\_t *mode* )

When disabling the TPM counter, the function will wait till it receives an acknowledge from the TPM clock domain

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>tpmBase</i> | TPM module base address pointer      |
| <i>mode</i>    | The TPM counter clock mode (source). |

### 38.2.4.5 static tpm\_clock\_mode\_t TPM\_HAL\_GetClockMode ( TPM\_Type \* *tpmBase* ) [inline], [static]

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
|----------------|---------------------------------|

Returns

The TPM counter clock mode (source).

### 38.2.4.6 static void TPM\_HAL\_SetClockDiv ( TPM\_Type \* *tpmBase*, tpm\_clock\_ps\_t *ps* ) [inline], [static]

## TPM HAL driver

### Parameters

|                |                                           |
|----------------|-------------------------------------------|
| <i>tpmBase</i> | TPM module base address pointer           |
| <i>ps</i>      | The TPM peripheral clock prescale divider |

### 38.2.4.7 `static tpm_clock_ps_t TPM_HAL_GetClockDiv ( TPM_Type * tpmBase )` `[inline], [static]`

### Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
|----------------|---------------------------------|

### Returns

The TPM peripheral clock prescale divider.

### 38.2.4.8 `static void TPM_HAL_EnableTimerOverflowInt ( TPM_Type * tpmBase )` `[inline], [static]`

### Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
|----------------|---------------------------------|

### 38.2.4.9 `static void TPM_HAL_DisableTimerOverflowInt ( TPM_Type * tpmBase )` `[inline], [static]`

### Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
|----------------|---------------------------------|

### 38.2.4.10 `static bool TPM_HAL_IsOverflowIntEnabled ( TPM_Type * tpmBase )` `[inline], [static]`

### Parameters

---

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
|----------------|---------------------------------|

Returns

true if overflow interrupt is enabled, false if not

**38.2.4.11 static bool TPM\_HAL\_GetTimerOverflowStatus ( TPM\_Type \* *tpmBase* )  
[inline], [static]**

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
|----------------|---------------------------------|

Returns

true if overflow, false if not

**38.2.4.12 static void TPM\_HAL\_ClearTimerOverflowFlag ( TPM\_Type \* *tpmBase* )  
[inline], [static]**

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
|----------------|---------------------------------|

**38.2.4.13 static void TPM\_HAL\_SetCpwms ( TPM\_Type \* *tpmBase*, uint8\_t *mode* )  
[inline], [static]**

Parameters

|                |                                            |
|----------------|--------------------------------------------|
| <i>tpmBase</i> | TPM module base address pointer            |
| <i>mode</i>    | 1:upcounting mode 0:up_down counting mode. |

**38.2.4.14 static bool TPM\_HAL\_GetCpwms ( TPM\_Type \* *tpmBase* ) [inline],  
[static]**

## TPM HAL driver

### Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
|----------------|---------------------------------|

### Returns

Whether the TPM center-aligned PWM is selected or not.

**38.2.4.15** `static void TPM_HAL_ClearCounter ( TPM_Type * tpmBase ) [inline], [static]`

### Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
|----------------|---------------------------------|

**38.2.4.16** `static uint16_t TPM_HAL_GetCounterVal ( TPM_Type * tpmBase ) [inline], [static]`

### Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
|----------------|---------------------------------|

### Returns

current TPM timer counter value

**38.2.4.17** `static void TPM_HAL_SetMod ( TPM_Type * tpmBase, uint16_t val ) [inline], [static]`

### Parameters

|                |                                         |
|----------------|-----------------------------------------|
| <i>tpmBase</i> | TPM module base address pointer         |
| <i>val</i>     | The value to be set to the timer modulo |

**38.2.4.18** `static uint16_t TPM_HAL_GetMod ( TPM_Type * tpmBase ) [inline], [static]`



Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
|----------------|---------------------------------|

Returns

TPM timer modula value

**38.2.4.19 static void TPM\_HAL\_SetChnMsnbaElsnbaVal ( TPM\_Type \* *tpmBase*, uint8\_t *channel*, uint8\_t *value* ) [inline], [static]**

TPM channel operate mode, MSnBA and ELSnBA should be set at the same time.

Parameters

|                |                                       |
|----------------|---------------------------------------|
| <i>tpmBase</i> | The TPM base address                  |
| <i>channel</i> | The TPM peripheral channel number     |
| <i>value</i>   | The value to set for MSnBA and ELSnBA |

**38.2.4.20 static uint8\_t TPM\_HAL\_GetChnMsnbaVal ( TPM\_Type \* *tpmBase*, uint8\_t *channel* ) [inline], [static]**

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>tpmBase</i> | TPM module base address pointer   |
| <i>channel</i> | The TPM peripheral channel number |

Returns

The MSnB:MSnA mode value, will be 00,01, 10, 11

**38.2.4.21 static uint8\_t TPM\_HAL\_GetChnElsnbaVal ( TPM\_Type \* *tpmBase*, uint8\_t *channel* ) [inline], [static]**

## TPM HAL driver

### Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>tpmBase</i> | TPM module base address pointer   |
| <i>channel</i> | The TPM peripheral channel number |

### Returns

The ELSnB:ELSnA mode value, will be 00,01, 10, 11

**38.2.4.22** `static void TPM_HAL_EnableChnInt ( TPM_Type * tpmBase, uint8_t channel )`  
**[inline], [static]**

### Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>tpmBase</i> | TPM module base address pointer   |
| <i>channel</i> | The TPM peripheral channel number |

**38.2.4.23** `static void TPM_HAL_DisableChnInt ( TPM_Type * tpmBase, uint8_t channel )`  
**[inline], [static]**

### Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>tpmBase</i> | TPM module base address pointer   |
| <i>channel</i> | The TPM peripheral channel number |

**38.2.4.24** `static bool TPM_HAL_IsChnIntEnabled ( TPM_Type * tpmBase, uint8_t channel )`  
**[inline], [static]**

### Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>tpmBase</i> | TPM module base address pointer   |
| <i>channel</i> | The TPM peripheral channel number |

### Returns

Whether the TPM peripheral timer channel(n) interrupt is enabled or not.

**38.2.4.25** `static bool TPM_HAL_GetChnStatus ( TPM_Type * tpmBase, uint8_t channel )`  
**[inline], [static]**

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>tpmBase</i> | TPM module base address pointer    |
| <i>channel</i> | The TPM peripheral channel number. |

Returns

true if event occurred, false otherwise

**38.2.4.26 static void TPM\_HAL\_ClearChnInt ( TPM\_Type \* *tpmBase*, uint8\_t *channel* ) [inline], [static]**

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>tpmBase</i> | TPM module base address pointer    |
| <i>channel</i> | The TPM peripheral channel number. |

**38.2.4.27 static void TPM\_HAL\_SetChnCountVal ( TPM\_Type \* *tpmBase*, uint8\_t *channel*, uint16\_t *val* ) [inline], [static]**

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>tpmBase</i> | TPM module base address pointer    |
| <i>channel</i> | The TPM peripheral channel number. |
| <i>val</i>     | counter value to be set            |

**38.2.4.28 static uint16\_t TPM\_HAL\_GetChnCountVal ( TPM\_Type \* *tpmBase*, uint8\_t *channel* ) [inline], [static]**

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>tpmBase</i> | TPM module base address pointer    |
| <i>channel</i> | The TPM peripheral channel number. |

Returns

The TPM timer channel counter value.

---

## TPM HAL driver

**38.2.4.29** `static uint32_t TPM_HAL_GetStatusRegVal ( TPM_Type * tpmBase )`  
`[inline], [static]`

## Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
|----------------|---------------------------------|

## Returns

The TPM timer channel event status.

**38.2.4.30** `static void TPM_HAL_ClearStatusReg ( TPM_Type * tpmBase, uint16_t tpm_status ) [inline], [static]`

## Parameters

|                   |                                       |
|-------------------|---------------------------------------|
| <i>tpmBase</i>    | TPM module base address pointer       |
| <i>tpm_status</i> | tpm channel or overflow flag to clear |

**38.2.4.31** `static void TPM_HAL_SetTriggerSrc ( TPM_Type * tpmBase, tpm_trigger_source_t trigger_num ) [inline], [static]`

## Parameters

|                    |                                 |
|--------------------|---------------------------------|
| <i>tpmBase</i>     | TPM module base address pointer |
| <i>trigger_num</i> | 0-15                            |

**38.2.4.32** `static void TPM_HAL_SetTriggerMode ( TPM_Type * tpmBase, bool enable ) [inline], [static]`

## Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
| <i>enable</i>  | true to enable, 1 to enable     |

**38.2.4.33** `static void TPM_HAL_SetReloadOnTriggerMode ( TPM_Type * tpmBase, bool enable ) [inline], [static]`

## TPM HAL driver

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>tpmBase</i> | TPM module base address pointer   |
| <i>enable</i>  | true to enable, false to disable. |

**38.2.4.34** `static void TPM_HAL_SetStopOnOverflowMode ( TPM_Type * tpmBase, bool enable ) [inline], [static]`

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>tpmBase</i> | TPM module base address pointer   |
| <i>enable</i>  | true to enable, false to disable. |

**38.2.4.35** `static void TPM_HAL_EnableGlobalTimeBase ( TPM_Type * tpmBase, bool enable ) [inline], [static]`

Parameters

|                |                                   |
|----------------|-----------------------------------|
| <i>tpmBase</i> | TPM module base address pointer   |
| <i>enable</i>  | true to enable, false to disable. |

**38.2.4.36** `static void TPM_HAL_SetDbgMode ( TPM_Type * tpmBase, bool enable ) [inline], [static]`

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>tpmBase</i> | TPM module base address pointer |
| <i>enable</i>  | false pause, true continue work |

**38.2.4.37** `static void TPM_HAL_SetWaitMode ( TPM_Type * tpmBase, bool enable ) [inline], [static]`

Parameters

---

|                |                                    |
|----------------|------------------------------------|
| <i>tpmBase</i> | TPM module base address pointer    |
| <i>enable</i>  | 0 continue running, 1 stop running |

## TPM Peripheral driver

### 38.3 TPM Peripheral driver

#### 38.3.1 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.

#### 38.3.2 TPM Initialization

1. To initialize the TPM driver, call the `TPM_DRV_Init()` function and pass the instance number of the relevant TPM. For example, 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. The user needs to set the relevant configurations.
typedef struct TpmGeneralConfig {
 bool isDBGMode;
 bool isGlobalTimeBase;
 bool isTriggerMode;
 bool isStopCountOnOverflow;
 bool isCountReloadOnTrig;
 tpm_trigger_source_t triggerSource;
} tpm_general_config_t;
```

#### 38.3.3 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 Hertz and the duty cycle percentage (value between 0-100).

#### 38.3.4 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:



```
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.

### 38.3.5 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:

```
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.

### 38.3.6 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...](#)*

## TPM Peripheral driver

### Enumerations

- enum `tpm_clock_source_t` {  
`kTpmClockSourceNone` = 0,  
`kTpmClockSourceModuleHighFreq`,  
`kTpmClockSourceModuleOSCERCLK`,  
`kTpmClockSourceModuleMCGIRCLK`,  
`kTpmClockSourceExternalCLKIN0`,  
`kTpmClockSourceExternalCLKIN1`,  
`kTpmClockSourceReserved` }

*TPM clock source selection.*

### Functions

- `tpm_status_t TPM_DRV_Init` (uint32\_t instance, const `tpm_general_config_t` \*info)  
*Initializes the TPM driver.*
- void `TPM_DRV_PwmStop` (uint32\_t instance, `tpm_pwm_param_t` \*param, uint8\_t channel)  
*Stops the channel PWM.*
- `tpm_status_t TPM_DRV_PwmStart` (uint32\_t instance, `tpm_pwm_param_t` \*param, uint8\_t channel)  
*Configures duty cycle and 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` (uint32\_t instance, `tpm_clock_source_t` clock, `tpm_clock_ps_t` clockPs)  
*Sets the TPM clock source.*
- uint32\_t `TPM_DRV_GetClock` (uint32\_t instance)  
*Gets the TPM clock frequency.*
- void `TPM_DRV_CounterStart` (uint32\_t instance, `tpm_counting_mode_t` countMode, uint32\_t countFinalVal, bool enableOverflowInt)  
*Starts the TPM counter.*
- void `TPM_DRV_CounterStop` (uint32\_t instance)  
*Stops the TPM counter.*
- uint32\_t `TPM_DRV_CounterRead` (uint32\_t instance)  
*Reads back the current value of the TPM counter.*
- void `TPM_DRV_InputCaptureEnable` (uint32\_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` (uint32\_t instance, uint8\_t channel)  
*Reads back the current value of the TPM channel value.*
- void `TPM_DRV_OutputCompareEnable` (uint32\_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` (uint32\_t instance)  
*Shuts down the TPM driver.*
- void `TPM_DRV_IRQHandler` (uint32\_t instance)  
*Action to take when an TPM interrupt is triggered.*

## Variables

- TPM\_Type \*const [g\\_tpmBase](#) [TPM\_INSTANCE\_COUNT]  
*Table of base addresses for TPM instances.*
- const IRQn\_Type [g\\_tpmIrqId](#) [TPM\_INSTANCE\_COUNT]  
*Table to save TPM IRQ numbers for TPM instances.*

## 38.3.7 Data Structure Documentation

### 38.3.7.1 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.*

## 38.3.8 Enumeration Type Documentation

### 38.3.8.1 enum tpm\_clock\_source\_t

Enumerator

***kTpmClockSourceNone*** TPM clock source, None.

***kTpmClockSourceModuleHighFreq*** TPM clock source, IRC48MHz or FLL/PLL depending on SoC.

***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.

## TPM Peripheral driver

### 38.3.9 Function Documentation

38.3.9.1 `tpm_status_t TPM_DRV_Init ( uint32_t instance, const tpm_general_config_t * info )`

Parameters

|                 |                                                                                             |
|-----------------|---------------------------------------------------------------------------------------------|
| <i>instance</i> | The TPM peripheral instance number.                                                         |
| <i>info</i>     | Pointer to the TPM user configuration structure, see <a href="#">tpm_general_config_t</a> . |

Returns

kStatusTpmSuccess means success, otherwise means failed.

**38.3.9.2 void TPM\_DRV\_PwmStop ( uint32\_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.                    |

**38.3.9.3 tpm\_status\_t TPM\_DRV\_PwmStart ( uint32\_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, see <a href="#">tpm_pwm_param_t</a> . |
| <i>channel</i>  | The channel number.                                                           |

Returns

kStatusTpmSuccess means success, otherwise means failed.

**38.3.9.4 void TPM\_DRV\_SetTimeOverflowIntCmd ( uint32\_t *instance*, bool *overflowEnable* )**

## TPM Peripheral driver

Parameters

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <i>instance</i>       | The TPM peripheral instance number.                       |
| <i>overflowEnable</i> | true: enable the timer overflow interrupt, false: disable |

### 38.3.9.5 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 |

### 38.3.9.6 void TPM\_DRV\_SetClock ( uint32\_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, see <a href="#">tpm_clock_source_t</a> .   |
| <i>clockPs</i>  | The TPM peripheral clock prescale factor, see <a href="#">tpm_clock_ps_t</a> . |

### 38.3.9.7 uint32\_t TPM\_DRV\_GetClock ( uint32\_t *instance* )

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The TPM peripheral instance number. |
|-----------------|-------------------------------------|

Returns

The function returns the frequency of the TPM clock.

### 38.3.9.8 void TPM\_DRV\_CounterStart ( uint32\_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 <code>tpm_counting_mode_t</code> . |
| <i>countFinalVal</i>      | The final value that is stored in the MOD register.                |
| <i>enable-OverflowInt</i> | true: enable timer overflow interrupt; false: disable              |

**38.3.9.9 void TPM\_DRV\_CounterStop ( uint32\_t instance )**

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The TPM peripheral instance number. |
|-----------------|-------------------------------------|

**38.3.9.10 uint32\_t TPM\_DRV\_CounterRead ( uint32\_t instance )**

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The TPM peripheral instance number. |
|-----------------|-------------------------------------|

Returns

The current value of the TPM counter.

**38.3.9.11 void TPM\_DRV\_InputCaptureEnable ( uint32\_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 <code>tpm_input_capture_mode_t</code> . |

## TPM Peripheral driver

|                      |                                                     |
|----------------------|-----------------------------------------------------|
| <i>countFinalVal</i> | The final value that is stored in the MOD register. |
| <i>intEnable</i>     | true: enable channel interrupt; false: disable      |

### 38.3.9.12 uint32\_t TPM\_DRV\_GetChnVal ( uint32\_t instance, uint8\_t channel )

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The TPM peripheral instance number. |
| <i>channel</i>  | The channel number.                 |

Returns

The current value of the TPM channel value.

### 38.3.9.13 void TPM\_DRV\_OutputCompareEnable ( uint32\_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 <a href="#">tpm_output_compare_mode_t</a> . |
| <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                             |

### 38.3.9.14 void TPM\_DRV\_Deinit ( uint32\_t instance )

Parameters

|                 |                                     |
|-----------------|-------------------------------------|
| <i>instance</i> | The TPM peripheral instance number. |
|-----------------|-------------------------------------|

### 38.3.9.15 void TPM\_DRV\_IRQHandler ( uint32\_t instance )

The timer overflow flag is checked and cleared if set.



Parameters

|                 |                                    |
|-----------------|------------------------------------|
| <i>instance</i> | Instance number of the TPM module. |
|-----------------|------------------------------------|

### 38.3.10 Variable Documentation

38.3.10.1 TPM\_Type\* const g\_tpmBase[TPM\_INSTANCE\_COUNT]

38.3.10.2 const IRQn\_Type g\_tpmlrqlId[TPM\_INSTANCE\_COUNT]



## Chapter 39

# Touch Sense Input (TSI)

### 39.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for Touch Sense Input drivers (TSI) block of Kinetis devices

#### Modules

- [TSI HAL driver](#)
- [TSI Peripheral Driver](#)

## TSI HAL driver

### 39.2 TSI HAL driver

#### 39.2.1 Overview

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

#### 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 }
  - enum [tsi\\_n\\_consecutive\\_scans\\_t](#) {
- Error codes for the TSI driver.*

```

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_2uA = 0,
kTsiRefOscChargeCurrent_4uA = 1,
kTsiRefOscChargeCurrent_6uA = 2,
kTsiRefOscChargeCurrent_8uA = 3,
kTsiRefOscChargeCurrent_10uA = 4,
kTsiRefOscChargeCurrent_12uA = 5,
kTsiRefOscChargeCurrent_14uA = 6,
kTsiRefOscChargeCurrent_16uA = 7,
kTsiRefOscChargeCurrent_18uA = 8,
kTsiRefOscChargeCurrent_20uA = 9,
kTsiRefOscChargeCurrent_22uA = 10,
kTsiRefOscChargeCurrent_24uA = 11,
kTsiRefOscChargeCurrent_26uA = 12,
kTsiRefOscChargeCurrent_28uA = 13,
kTsiRefOscChargeCurrent_30uA = 14,
kTsiRefOscChargeCurrent_32uA = 15,
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` {

```

kTsiExtOscChargeCurrent_2uA = 0,
kTsiExtOscChargeCurrent_4uA = 1,
kTsiExtOscChargeCurrent_6uA = 2,
kTsiExtOscChargeCurrent_8uA = 3,
kTsiExtOscChargeCurrent_10uA = 4,
kTsiExtOscChargeCurrent_12uA = 5,
kTsiExtOscChargeCurrent_14uA = 6,
kTsiExtOscChargeCurrent_16uA = 7,
kTsiExtOscChargeCurrent_18uA = 8,
kTsiExtOscChargeCurrent_20uA = 9,
kTsiExtOscChargeCurrent_22uA = 10,
kTsiExtOscChargeCurrent_24uA = 11,
kTsiExtOscChargeCurrent_26uA = 12,
kTsiExtOscChargeCurrent_28uA = 13,
kTsiExtOscChargeCurrent_30uA = 14,
kTsiExtOscChargeCurrent_32uA = 15,
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.*

## TSI HAL driver

- 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` {



```

kTsiRefOscChargeCurrent_2uA = 0,
kTsiRefOscChargeCurrent_4uA = 1,
kTsiRefOscChargeCurrent_6uA = 2,
kTsiRefOscChargeCurrent_8uA = 3,
kTsiRefOscChargeCurrent_10uA = 4,
kTsiRefOscChargeCurrent_12uA = 5,
kTsiRefOscChargeCurrent_14uA = 6,
kTsiRefOscChargeCurrent_16uA = 7,
kTsiRefOscChargeCurrent_18uA = 8,
kTsiRefOscChargeCurrent_20uA = 9,
kTsiRefOscChargeCurrent_22uA = 10,
kTsiRefOscChargeCurrent_24uA = 11,
kTsiRefOscChargeCurrent_26uA = 12,
kTsiRefOscChargeCurrent_28uA = 13,
kTsiRefOscChargeCurrent_30uA = 14,
kTsiRefOscChargeCurrent_32uA = 15,
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_oscilator_voltage_rails_t` {
 

```

kTsiOscVolRails_Dv_103 = 0,
kTsiOscVolRails_Dv_073 = 1,
kTsiOscVolRails_Dv_043 = 2,
kTsiOscVolRails_Dv_029 = 3 }

```
- enum `tsi_external_osc_charge_current_t` {

*TSI oscilator's voltage rails.*

## TSI HAL driver

```

kTsiExtOscChargeCurrent_2uA = 0,
kTsiExtOscChargeCurrent_4uA = 1,
kTsiExtOscChargeCurrent_6uA = 2,
kTsiExtOscChargeCurrent_8uA = 3,
kTsiExtOscChargeCurrent_10uA = 4,
kTsiExtOscChargeCurrent_12uA = 5,
kTsiExtOscChargeCurrent_14uA = 6,
kTsiExtOscChargeCurrent_16uA = 7,
kTsiExtOscChargeCurrent_18uA = 8,
kTsiExtOscChargeCurrent_20uA = 9,
kTsiExtOscChargeCurrent_22uA = 10,
kTsiExtOscChargeCurrent_24uA = 11,
kTsiExtOscChargeCurrent_26uA = 12,
kTsiExtOscChargeCurrent_28uA = 13,
kTsiExtOscChargeCurrent_30uA = 14,
kTsiExtOscChargeCurrent_32uA = 15,
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 {

```

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** (TSI\_Type \*base)  
*Initialize hardware.*
- void **TSI\_HAL\_SetConfiguration** (TSI\_Type \*base, tsi\_config\_t \*config)  
*Set configuration of hardware.*
- uint32\_t **TSI\_HAL\_Recalibrate** (TSI\_Type \*base, tsi\_config\_t \*config, const uint32\_t electrodes, const tsi\_parameter\_limits\_t \*parLimits)  
*Recalibrate TSI hardware.*
- void **TSI\_HAL\_EnableLowPower** (TSI\_Type \*base)  
*Enable low power for TSI module.*
- void **TSI\_HAL\_DisableLowPower** (TSI\_Type \*base)  
*Disable low power for TSI module.*
- static uint32\_t **TSI\_HAL\_IsModuleEnabled** (TSI\_Type \*base)  
*Get module flag enable.*
- static uint32\_t **TSI\_HAL\_GetScanTriggerMode** (TSI\_Type \*base)  
*Get TSI scan trigger mode.*
- static uint32\_t **TSI\_HAL\_IsScanInProgress** (TSI\_Type \*base)  
*Get scan in progress flag.*
- static uint32\_t **TSI\_HAL\_GetEndOfScanFlag** (TSI\_Type \*base)  
*Get end of scan flag.*
- static uint32\_t **TSI\_HAL\_GetOutOfRangeFlag** (TSI\_Type \*base)  
*Get out of range flag.*
- static  
**tsi\_electrode\_osc\_prescaler\_t** **TSI\_HAL\_GetPrescaler** (TSI\_Type \*base)  
*Get prescaler.*
- static **tsi\_n\_consecutive\_scans\_t** **TSI\_HAL\_GetNumberOfScans** (TSI\_Type \*base)  
*Get number of scans (NSCN).*
- static void **TSI\_HAL\_EnableModule** (TSI\_Type \*base)  
*Enable Touch Sensing Input Module.*
- static void **TSI\_HAL\_DisableModule** (TSI\_Type \*base)  
*Disable Touch Sensing Input Module.*
- static void **TSI\_HAL\_EnableStop** (TSI\_Type \*base)  
*Enable TSI module in stop mode.*
- static void **TSI\_HAL\_DisableStop** (TSI\_Type \*base)  
*Disable TSI module in stop mode.*
- static void **TSI\_HAL\_EnableOutOfRangeInterrupt** (TSI\_Type \*base)  
*Enable out of range interrupt.*
- static void **TSI\_HAL\_EnableEndOfScanInterrupt** (TSI\_Type \*base)  
*Enable end of scan interrupt.*
- static void **TSI\_HAL\_EnablePeriodicalScan** (TSI\_Type \*base)  
*Enable periodical (hardware) trigger scan.*
- static void **TSI\_HAL\_EnableSoftwareTriggerScan** (TSI\_Type \*base)  
*Enable software trigger scan.*
- static void **TSI\_HAL\_EnableErrorInterrupt** (TSI\_Type \*base)  
*Enable error interrupt.*
- static void **TSI\_HAL\_DisableErrorInterrupt** (TSI\_Type \*base)  
*Disable error interrupt.*
- static void **TSI\_HAL\_ClearOutOfRangeFlag** (TSI\_Type \*base)  
*Clear out of range flag.*
- static void **TSI\_HAL\_ClearEndOfScanFlag** (TSI\_Type \*base)

## TSI HAL driver

- Clear end of scan flag.*

  - static void [TSI\\_HAL\\_EnableInterrupt](#) (TSI\_Type \*base)

*Enable TSI module interrupt.*
- static void [TSI\\_HAL\\_DisableInterrupt](#) (TSI\_Type \*base)

*Disable TSI interrupt.*
- static uint32\_t [TSI\\_HAL\\_IsInterruptEnabled](#) (TSI\_Type \*base)

*Get interrupt enable flag.*
- static void [TSI\\_HAL\\_StartSoftwareTrigger](#) (TSI\_Type \*base)

*Start measurement (trigger the new measurement).*
- static uint32\_t [TSI\\_HAL\\_IsOverrun](#) (TSI\_Type \*base)

*Get overrun flag.*
- static void [TSI\\_HAL\\_ClearOverrunFlag](#) (TSI\_Type \*base)

*Clear over run flag.*
- static uint32\_t [TSI\\_HAL\\_GetExternalElectrodeErrorFlag](#) (TSI\_Type \*base)

*Get external electrode error flag.*
- static void [TSI\\_HAL\\_ClearExternalElectrodeErrorFlag](#) (TSI\_Type \*base)

*Clear external electrode error flag.*
- static void [TSI\\_HAL\\_SetPrescaler](#) (TSI\_Type \*base, [tsi\\_electrode\\_osc\\_prescaler\\_t](#) prescaler)

*Set prescaler.*
- static void [TSI\\_HAL\\_SetNumberOfScans](#) (TSI\_Type \*base, [tsi\\_n\\_consecutive\\_scans\\_t](#) number)

*Set number of scans (NSCN).*
- static void [TSI\\_HAL\\_SetLowPowerScanInterval](#) (TSI\_Type \*base, [tsi\\_low\\_power\\_interval\\_t](#) interval)

*Set low power scan interval.*
- static [tsi\\_low\\_power\\_interval\\_t](#) [TSI\\_HAL\\_GetLowPowerScanInterval](#) (TSI\_Type \*base)

*Get low power scan interval.*
- static void [TSI\\_HAL\\_SetLowPowerClock](#) (TSI\_Type \*base, uint32\_t clock)

*Set low power clock.*
- static uint32\_t [TSI\\_HAL\\_GetLowPowerClock](#) (TSI\_Type \*base)

*Get low power clock.*
- static void [TSI\\_HAL\\_SetReferenceChargeCurrent](#) (TSI\_Type \*base, [tsi\\_reference\\_osc\\_charge\\_current\\_t](#) current)

*Set the reference oscillator charge current.*
- static [tsi\\_reference\\_osc\\_charge\\_current\\_t](#) [TSI\\_HAL\\_GetReferenceChargeCurrent](#) (TSI\_Type \*base)

*Get the reference oscillator charge current.*
- static void [TSI\\_HAL\\_SetElectrodeChargeCurrent](#) (TSI\_Type \*base, [tsi\\_external\\_osc\\_charge\\_current\\_t](#) current)

*Set electrode charge current.*
- static [tsi\\_external\\_osc\\_charge\\_current\\_t](#) [TSI\\_HAL\\_GetElectrodeChargeCurrent](#) (TSI\_Type \*base)

*Get electrode charge current.*
- static void [TSI\\_HAL\\_SetScanModulo](#) (TSI\_Type \*base, uint32\_t modulo)

*Set scan modulo value.*
- static uint32\_t [TSI\\_HAL\\_GetScanModulo](#) (TSI\_Type \*base)

*Get scan modulo value.*
- static void [TSI\\_HAL\\_SetActiveModeSource](#) (TSI\_Type \*base, uint32\_t source)

*Set active mode source.*
- static uint32\_t [TSI\\_HAL\\_GetActiveModeSource](#) (TSI\_Type \*base)

*Get active mode source.*
- static void [TSI\\_HAL\\_SetActiveModePrescaler](#) (TSI\_Type \*base, [tsi\\_active\\_mode\\_prescaler\\_t](#)

- prescaler)
- *Set active mode prescaler.*  
static uint32\_t [TSI\\_HAL\\_GetActiveModePrescaler](#) (TSI\_Type \*base)
  - *Get active mode prescaler.*  
static void [TSI\\_HAL\\_SetLowPowerChannel](#) (TSI\_Type \*base, uint32\_t channel)
  - *Set low power channel.*  
static uint32\_t [TSI\\_HAL\\_GetLowPowerChannel](#) (TSI\_Type \*base)
  - *Get low power channel.*  
static void [TSI\\_HAL\\_EnableChannel](#) (TSI\_Type \*base, uint32\_t channel)
  - *Enable channel.*  
static void [TSI\\_HAL\\_EnableChannels](#) (TSI\_Type \*base, uint32\_t channelsMask)
  - *Enable channels.*  
static void [TSI\\_HAL\\_DisableChannel](#) (TSI\_Type \*base, uint32\_t channel)
  - *Disable channel.*  
static void [TSI\\_HAL\\_DisableChannels](#) (TSI\_Type \*base, uint32\_t channelsMask)
  - *Disable channels.*  
static uint32\_t [TSI\\_HAL\\_GetEnabledChannel](#) (TSI\_Type \*base, uint32\_t channel)
  - *Returns if channel is enabled.*  
static uint32\_t [TSI\\_HAL\\_GetEnabledChannels](#) (TSI\_Type \*base)
  - *Returns mask of enabled channels.*  
static uint16\_t [TSI\\_HAL\\_GetWakeUpChannelCounter](#) (TSI\_Type \*base)
  - *Set the Wake up channel counter.*  
static uint32\_t [TSI\\_HAL\\_GetCounter](#) (TSI\_Type \*base, uint32\_t channel)
  - *Get tsi counter on actual channel.*  
static void [TSI\\_HAL\\_SetLowThreshold](#) (TSI\_Type \*base, uint32\_t low\_threshold)
  - *Set low threshold.*  
static void [TSI\\_HAL\\_SetHighThreshold](#) (TSI\_Type \*base, uint32\_t high\_threshold)
  - *Set high threshold.*  
static uint32\_t [TSI\\_HAL\\_GetEnableStop](#) (TSI\_Type \*base)
  - *Get TSI STOP enable.*  
static void [TSI\\_HAL\\_EnableHardwareTriggerScan](#) (TSI\_Type \*base)
  - *Enable periodical (hardware) trigger scan.*  
static void [TSI\\_HAL\\_CurrentSourcePairSwapped](#) (TSI\_Type \*base)
  - *The current sources (CURSW) of electrode oscillator and reference oscillator are swapped.*  
static void [TSI\\_HAL\\_CurrentSourcePairNotSwapped](#) (TSI\_Type \*base)
  - *The current sources (CURSW) of electrode oscillator and reference oscillator are not swapped.*  
static uint32\_t [TSI\\_HAL\\_GetCurrentSourcePairSwapped](#) (TSI\_Type \*base)
  - *Get current source pair swapped status.*  
static void [TSI\\_HAL\\_SetMeasuredChannelNumber](#) (TSI\_Type \*base, uint32\_t channel)
  - *Set the measured channel number.*  
static uint32\_t [TSI\\_HAL\\_GetMeasuredChannelNumber](#) (TSI\_Type \*base)
  - *Get the measured channel number.*  
static void [TSI\\_HAL\\_DmaTransferEnable](#) (TSI\_Type \*base)
  - *DMA transfer enable.*  
static void [TSI\\_HAL\\_DmaTransferDisable](#) (TSI\_Type \*base)
  - *DMA transfer disable - do not generate DMA transfer request.*  
static uint32\_t [TSI\\_HAL\\_IsDmaTransferEnable](#) (TSI\_Type \*base)
  - *Get DMA transfer enable flag.*  
static uint32\_t [TSI\\_HAL\\_GetCounter](#) (TSI\_Type \*base)
  - *Get conversion counter value.*  
static void [TSI\\_HAL\\_SetMode](#) (TSI\_Type \*base, [tsi\\_analog\\_mode\\_select\\_t](#) mode)

## TSI HAL driver

- *Set analog mode of the TSI module.*
- static [tsi\\_analog\\_mode\\_select\\_t TSI\\_HAL\\_GetMode](#) (TSI\_Type \*base)
- *Get analog mode of the TSI module.*
- static [uint32\\_t TSI\\_HAL\\_GetNoiseResult](#) (TSI\_Type \*base)
- *Get analog mode of the TSI module.*
- static void [TSI\\_HAL\\_SetOscillatorVoltageRails](#) (TSI\_Type \*base, [tsi\\_oscilator\\_voltage\\_rails\\_t](#) dvolt)
- *Set the oscillator's volatage rails.*
- static [tsi\\_oscilator\\_voltage\\_rails\\_t TSI\\_HAL\\_GetOscillatorVoltageRails](#) (TSI\_Type \*base)
- *Get the oscillator's volatage rails.*

## 39.2.2 Data Structure Documentation

### 39.2.2.1 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*

### 39.2.2.2 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.*

### 39.2.2.3 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.*

#### 39.2.2.4 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.*

#### 39.2.2.5 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.*

#### 39.2.2.6 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 HAL driver

- *TSI mode of operation.*
- [tsi\\_oscilator\\_voltage\\_rails\\_t](#) *dvolt*  
*Oscillator's voltage rails.*
- [uint16\\_t thresh](#)  
*High threshold.*
- [uint16\\_t thresl](#)  
*Low threshold.*

### 39.2.2.6.0.67 Field Documentation

**39.2.2.6.0.67.1** [tsi\\_n\\_consecutive\\_scans\\_t](#) [tsi\\_config\\_t::nscn](#)

**39.2.2.6.0.67.2** [tsi\\_analog\\_mode\\_select\\_t](#) [tsi\\_config\\_t::mode](#)

**39.2.2.6.0.67.3** [tsi\\_oscilator\\_voltage\\_rails\\_t](#) [tsi\\_config\\_t::dvolt](#)

**39.2.2.6.0.67.4** [uint16\\_t](#) [tsi\\_config\\_t::thresh](#)

**39.2.2.6.0.67.5** [uint16\\_t](#) [tsi\\_config\\_t::thresl](#)

## 39.2.3 Enumeration Type Documentation

### 39.2.3.1 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.

### 39.2.3.2 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



|                                          |                           |
|------------------------------------------|---------------------------|
| <i>kTsiConsecutiveScansNumber_8time</i>  | 8 times consecutive scan  |
| <i>kTsiConsecutiveScansNumber_9time</i>  | 9 times consecutive scan  |
| <i>kTsiConsecutiveScansNumber_10time</i> | 10 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_11time</i> | 11 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_12time</i> | 12 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_13time</i> | 13 times consecutive scan |
| <i>kTsiConsecutiveScansNumber_14time</i> | 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 |

### 39.2.3.3 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. |

### 39.2.3.4 enum tsi\_low\_power\_interval\_t

These constants define the tsi low power scan intervals in a TSI instance.

## TSI HAL driver

### 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 |

### 39.2.3.5 enum tsi\_reference\_osc\_charge\_current\_t

These constants define the tsi Reference oscillator charge current select in a TSI instance.

### Enumerator

|                                      |                                               |
|--------------------------------------|-----------------------------------------------|
| <i>kTsiRefOscChargeCurrent_2uA</i>   | Reference oscillator charge current is 2uA.   |
| <i>kTsiRefOscChargeCurrent_4uA</i>   | Reference oscillator charge current is 4uA.   |
| <i>kTsiRefOscChargeCurrent_6uA</i>   | Reference oscillator charge current is 6uA.   |
| <i>kTsiRefOscChargeCurrent_8uA</i>   | Reference oscillator charge current is 8uA.   |
| <i>kTsiRefOscChargeCurrent_10uA</i>  | Reference oscillator charge current is 10uA.  |
| <i>kTsiRefOscChargeCurrent_12uA</i>  | Reference oscillator charge current is 12uA.  |
| <i>kTsiRefOscChargeCurrent_14uA</i>  | Reference oscillator charge current is 14uA.  |
| <i>kTsiRefOscChargeCurrent_16uA</i>  | Reference oscillator charge current is 16uA.  |
| <i>kTsiRefOscChargeCurrent_18uA</i>  | Reference oscillator charge current is 18uA.  |
| <i>kTsiRefOscChargeCurrent_20uA</i>  | Reference oscillator charge current is 20uA.  |
| <i>kTsiRefOscChargeCurrent_22uA</i>  | Reference oscillator charge current is 22uA.  |
| <i>kTsiRefOscChargeCurrent_24uA</i>  | Reference oscillator charge current is 24uA.  |
| <i>kTsiRefOscChargeCurrent_26uA</i>  | Reference oscillator charge current is 26uA.  |
| <i>kTsiRefOscChargeCurrent_28uA</i>  | Reference oscillator charge current is 28uA.  |
| <i>kTsiRefOscChargeCurrent_30uA</i>  | Reference oscillator charge current is 30uA.  |
| <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.   |

- kTsiRefOscChargeCurrent\_8uA* Reference oscillator charge current is 8uA.
- kTsiRefOscChargeCurrent\_16uA* Reference oscillator charge current is 16uA.
- kTsiRefOscChargeCurrent\_32uA* Reference oscillator charge current is 32uA.
- kTsiRefOscChargeCurrent\_64uA* Reference oscillator charge current is 64uA.

### 39.2.3.6 enum tsi\_external\_osc\_charge\_current\_t

These constants define the tsi External oscillator charge current select in a TSI instance.

Enumerator

- kTsiExtOscChargeCurrent\_2uA* External oscillator charge current is 2uA.
- kTsiExtOscChargeCurrent\_4uA* External oscillator charge current is 4uA.
- kTsiExtOscChargeCurrent\_6uA* External oscillator charge current is 6uA.
- kTsiExtOscChargeCurrent\_8uA* External oscillator charge current is 8uA.
- kTsiExtOscChargeCurrent\_10uA* External oscillator charge current is 10uA.
- kTsiExtOscChargeCurrent\_12uA* External oscillator charge current is 12uA.
- kTsiExtOscChargeCurrent\_14uA* External oscillator charge current is 14uA.
- kTsiExtOscChargeCurrent\_16uA* External oscillator charge current is 16uA.
- kTsiExtOscChargeCurrent\_18uA* External oscillator charge current is 18uA.
- kTsiExtOscChargeCurrent\_20uA* External oscillator charge current is 20uA.
- kTsiExtOscChargeCurrent\_22uA* External oscillator charge current is 22uA.
- kTsiExtOscChargeCurrent\_24uA* External oscillator charge current is 24uA.
- kTsiExtOscChargeCurrent\_26uA* External oscillator charge current is 26uA.
- kTsiExtOscChargeCurrent\_28uA* External oscillator charge current is 28uA.
- kTsiExtOscChargeCurrent\_30uA* External oscillator charge current is 30uA.
- kTsiExtOscChargeCurrent\_32uA* External oscillator charge current is 32uA.
- kTsiExtOscChargeCurrent\_500nA* External oscillator charge current is 500nA.
- kTsiExtOscChargeCurrent\_1uA* External oscillator charge current is 1uA.
- kTsiExtOscChargeCurrent\_2uA* External oscillator charge current is 2uA.
- kTsiExtOscChargeCurrent\_4uA* External oscillator charge current is 4uA.
- kTsiExtOscChargeCurrent\_8uA* External oscillator charge current is 8uA.
- kTsiExtOscChargeCurrent\_16uA* External oscillator charge current is 16uA.
- kTsiExtOscChargeCurrent\_32uA* External oscillator charge current is 32uA.
- kTsiExtOscChargeCurrent\_64uA* External oscillator charge current is 64uA.

### 39.2.3.7 enum tsi\_internal\_cap\_trim\_t

These constants define the tsi Internal capacitance trim value in a TSI instance.

Enumerator

- kTsiIntCapTrim\_0\_5pF* 0.5 pF internal reference capacitance
- kTsiIntCapTrim\_0\_6pF* 0.6 pF internal reference capacitance

## TSI HAL driver

|                             |                                       |
|-----------------------------|---------------------------------------|
| <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 |

### 39.2.3.8 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 |
| <i>kTsiOscDeltaVoltage_300mV</i> | 300 mV delta voltage is applied |
| <i>kTsiOscDeltaVoltage_400mV</i> | 400 mV delta voltage is applied |
| <i>kTsiOscDeltaVoltage_500mV</i> | 500 mV delta voltage is applied |
| <i>kTsiOscDeltaVoltage_600mV</i> | 600 mV delta voltage is applied |

### 39.2.3.9 enum tsi\_active\_mode\_clock\_divider\_t

These constants define the active mode clock divider in a TSI instance.

Enumerator

|                                 |                                           |
|---------------------------------|-------------------------------------------|
| <i>kTsiActiveClkDiv_1div</i>    | Active mode clock divider is set to 1.    |
| <i>kTsiActiveClkDiv_2048div</i> | Active mode clock divider is set to 2048. |

### 39.2.3.10 enum tsi\_active\_mode\_clock\_source\_t

These constants define the active mode clock source in a TSI instance.

Enumerator

|                                     |                                                                  |
|-------------------------------------|------------------------------------------------------------------|
| <i>kTsiActiveClkSource_BusClock</i> | Active mode clock source is set to Bus Clock.                    |
| <i>kTsiActiveClkSource_MCGIRCLK</i> | Active mode clock source is set to MCG Internal reference clock. |
| <i>kTsiActiveClkSource_OSCERCLK</i> | Active mode clock source is set to System oscillator output.     |

### 39.2.3.11 enum tsi\_active\_mode\_prescaler\_t

These constants define the tsi active mode prescaler in a TSI instance.

Enumerator

|                                       |                                    |
|---------------------------------------|------------------------------------|
| <i>kTsiActiveModePrescaler_1div</i>   | Input clock source divided by 1.   |
| <i>kTsiActiveModePrescaler_2div</i>   | Input clock source divided by 2.   |
| <i>kTsiActiveModePrescaler_4div</i>   | Input clock source divided by 4.   |
| <i>kTsiActiveModePrescaler_8div</i>   | Input clock source divided by 8.   |
| <i>kTsiActiveModePrescaler_16div</i>  | Input clock source divided by 16.  |
| <i>kTsiActiveModePrescaler_32div</i>  | Input clock source divided by 32.  |
| <i>kTsiActiveModePrescaler_64div</i>  | Input clock source divided by 64.  |
| <i>kTsiActiveModePrescaler_128div</i> | Input clock source divided by 128. |

### 39.2.3.12 enum tsi\_analog\_mode\_select\_t

Set up TSI analog modes in a TSI instance.

Enumerator

|                                         |                                                                                               |
|-----------------------------------------|-----------------------------------------------------------------------------------------------|
| <i>kTsiAnalogModeSel_Capacitive</i>     | Active TSI capacitive sensing mode.                                                           |
| <i>kTsiAnalogModeSel_NoiseNoFreqLim</i> | TSI works in single threshold noise detection mode and the freq. limitation is disabled       |
| <i>kTsiAnalogModeSel_NoiseFreqLim</i>   | TSI analog works in single threshold noise detection mode and the freq. limitation is enabled |

### 39.2.3.13 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_2uA</i>  | Reference oscillator charge current is 2uA.  |
| <i>kTsiRefOscChargeCurrent_4uA</i>  | Reference oscillator charge current is 4uA.  |
| <i>kTsiRefOscChargeCurrent_6uA</i>  | Reference oscillator charge current is 6uA.  |
| <i>kTsiRefOscChargeCurrent_8uA</i>  | Reference oscillator charge current is 8uA.  |
| <i>kTsiRefOscChargeCurrent_10uA</i> | Reference oscillator charge current is 10uA. |
| <i>kTsiRefOscChargeCurrent_12uA</i> | Reference oscillator charge current is 12uA. |
| <i>kTsiRefOscChargeCurrent_14uA</i> | Reference oscillator charge current is 14uA. |
| <i>kTsiRefOscChargeCurrent_16uA</i> | Reference oscillator charge current is 16uA. |
| <i>kTsiRefOscChargeCurrent_18uA</i> | Reference oscillator charge current is 18uA. |
| <i>kTsiRefOscChargeCurrent_20uA</i> | Reference oscillator charge current is 20uA. |
| <i>kTsiRefOscChargeCurrent_22uA</i> | Reference oscillator charge current is 22uA. |

## TSI HAL driver

|                                      |                                               |
|--------------------------------------|-----------------------------------------------|
| <i>kTsiRefOscChargeCurrent_24uA</i>  | Reference oscillator charge current is 24uA.  |
| <i>kTsiRefOscChargeCurrent_26uA</i>  | Reference oscillator charge current is 26uA.  |
| <i>kTsiRefOscChargeCurrent_28uA</i>  | Reference oscillator charge current is 28uA.  |
| <i>kTsiRefOscChargeCurrent_30uA</i>  | Reference oscillator charge current is 30uA.  |
| <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.  |

### 39.2.3.14 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. |

### 39.2.3.15 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_2uA</i>  | External oscillator charge current is 2uA.  |
| <i>kTsiExtOscChargeCurrent_4uA</i>  | External oscillator charge current is 4uA.  |
| <i>kTsiExtOscChargeCurrent_6uA</i>  | External oscillator charge current is 6uA.  |
| <i>kTsiExtOscChargeCurrent_8uA</i>  | External oscillator charge current is 8uA.  |
| <i>kTsiExtOscChargeCurrent_10uA</i> | External oscillator charge current is 10uA. |
| <i>kTsiExtOscChargeCurrent_12uA</i> | External oscillator charge current is 12uA. |
| <i>kTsiExtOscChargeCurrent_14uA</i> | External oscillator charge current is 14uA. |
| <i>kTsiExtOscChargeCurrent_16uA</i> | External oscillator charge current is 16uA. |
| <i>kTsiExtOscChargeCurrent_18uA</i> | External oscillator charge current is 18uA. |
| <i>kTsiExtOscChargeCurrent_20uA</i> | External oscillator charge current is 20uA. |
| <i>kTsiExtOscChargeCurrent_22uA</i> | External oscillator charge current is 22uA. |
| <i>kTsiExtOscChargeCurrent_24uA</i> | External oscillator charge current is 24uA. |

- kTsiExtOscChargeCurrent\_26uA* External oscillator charge current is 26uA.
- kTsiExtOscChargeCurrent\_28uA* External oscillator charge current is 28uA.
- kTsiExtOscChargeCurrent\_30uA* External oscillator charge current is 30uA.
- kTsiExtOscChargeCurrent\_32uA* External oscillator charge current is 32uA.
- kTsiExtOscChargeCurrent\_500nA* External oscillator charge current is 500nA.
- kTsiExtOscChargeCurrent\_1uA* External oscillator charge current is 1uA.
- kTsiExtOscChargeCurrent\_2uA* External oscillator charge current is 2uA.
- kTsiExtOscChargeCurrent\_4uA* External oscillator charge current is 4uA.
- kTsiExtOscChargeCurrent\_8uA* External oscillator charge current is 8uA.
- kTsiExtOscChargeCurrent\_16uA* External oscillator charge current is 16uA.
- kTsiExtOscChargeCurrent\_32uA* External oscillator charge current is 32uA.
- kTsiExtOscChargeCurrent\_64uA* External oscillator charge current is 64uA.

### 39.2.3.16 enum tsi\_channel\_number\_t

These bits specify current channel to be measured.

Enumerator

- kTsiChannelNumber\_0* Channel Number 0.
- kTsiChannelNumber\_1* Channel Number 1.
- kTsiChannelNumber\_2* Channel Number 2.
- kTsiChannelNumber\_3* Channel Number 3.
- kTsiChannelNumber\_4* Channel Number 4.
- kTsiChannelNumber\_5* Channel Number 5.
- kTsiChannelNumber\_6* Channel Number 6.
- kTsiChannelNumber\_7* Channel Number 7.
- kTsiChannelNumber\_8* 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.

## 39.2.4 Function Documentation

### 39.2.4.1 void TSI\_HAL\_Init ( TSI\_Type \* *base* )

## TSI HAL driver

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

none

Initialize the peripheral to default state.

### 39.2.4.2 void TSI\_HAL\_SetConfiguration ( TSI\_Type \* *base*, tsi\_config\_t \* *config* )

Parameters

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

### 39.2.4.3 uint32\_t TSI\_HAL\_Recalibrate ( TSI\_Type \* *base*, tsi\_config\_t \* *config*, const uint32\_t *electrodes*, const tsi\_parameter\_limits\_t \* *parLimits* )

Parameters

|                   |                                                   |
|-------------------|---------------------------------------------------|
| <i>base</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 implimentation of this function for better understanding [TSI\\_HAL\\_Recalibrate](#).



39.2.4.4 void TSI\_HAL\_EnableLowPower ( TSI\_Type \* *base* )

## TSI HAL driver

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

none

### 39.2.4.5 void TSI\_HAL\_DisableLowPower ( TSI\_Type \* *base* )

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

None.

### 39.2.4.6 static uint32\_t TSI\_HAL\_IsModuleEnabled ( TSI\_Type \* *base* ) [inline], [static]

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

State of enable module flag.

### 39.2.4.7 static uint32\_t TSI\_HAL\_GetScanTriggerMode ( TSI\_Type \* *base* ) [inline], [static]

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

Scan trigger mode.

### 39.2.4.8 static uint32\_t TSI\_HAL\_IsScanInProgress ( TSI\_Type \* *base* ) [inline], [static]

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

True - if scan is in progress. False - otherwise

**39.2.4.9 static uint32\_t TSI\_HAL\_GetEndOfScanFlag ( TSI\_Type \* *base* ) [inline], [static]**

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

Current state of end of scan flag.

**39.2.4.10 static uint32\_t TSI\_HAL\_GetOutOfRangeFlag ( TSI\_Type \* *base* ) [inline], [static]**

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

State of out of range flag.

**39.2.4.11 static tsi\_electrode\_osc\_prescaler\_t TSI\_HAL\_GetPrescaler ( TSI\_Type \* *base* ) [inline], [static]**

Parameters

---

## TSI HAL driver

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

Prescaler value.

**39.2.4.12** `static tsi_n_consecutive_scans_t TSI_HAL_GetNumberOfScans ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

Number of scans.

**39.2.4.13** `static void TSI_HAL_EnableModule ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.14** `static void TSI_HAL_DisableModule ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.15** `static void TSI_HAL_EnableStop ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.16 static void TSI\_HAL\_DisableStop ( TSI\_Type \* *base* ) [inline], [static]**

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.17 static void TSI\_HAL\_EnableOutOfRangeInterrupt ( TSI\_Type \* *base* ) [inline], [static]**

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.18 static void TSI\_HAL\_EnableEndOfScanInterrupt ( TSI\_Type \* *base* ) [inline], [static]**

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.19 static void TSI\_HAL\_EnablePeriodicalScan ( TSI\_Type \* *base* ) [inline], [static]**

## TSI HAL driver

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.20 static void TSI\_HAL\_EnableSoftwareTriggerScan ( TSI\_Type \* *base* )  
[inline], [static]**

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.21 static void TSI\_HAL\_EnableErrorInterrupt ( TSI\_Type \* *base* ) [inline],  
[static]**

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.22 static void TSI\_HAL\_DisableErrorInterrupt ( TSI\_Type \* *base* ) [inline],  
[static]**

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

39.2.4.23 `static void TSI_HAL_ClearOutOfRangeFlag ( TSI_Type * base ) [inline],  
[static]`

## TSI HAL driver

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.24** `static void TSI_HAL_ClearEndOfScanFlag ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.25** `static void TSI_HAL_EnableInterrupt ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.26** `static void TSI_HAL_DisableInterrupt ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.



```
39.2.4.27 static uint32_t TSI_HAL_IsInterruptEnabled (TSI_Type * base) [inline],
 [static]
```

## TSI HAL driver

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

State of enable interrupt flag.

**39.2.4.28** `static void TSI_HAL_StartSoftwareTrigger ( TSI_Type * base ) [inline], [static]`

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

None.

**39.2.4.29** `static uint32_t TSI_HAL_IsOverrun ( TSI_Type * base ) [inline], [static]`

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

State of over run flag.

**39.2.4.30** `static void TSI_HAL_ClearOverrunFlag ( TSI_Type * base ) [inline], [static]`

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

None.

**39.2.4.31** `static uint32_t TSI_HAL_GetExternalElectrodeErrorFlag ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

State of external electrode error flag

**39.2.4.32 static void TSI\_HAL\_ClearExternalElectrodeErrorFlag ( TSI\_Type \* *base* )  
[inline], [static]**

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.33 static void TSI\_HAL\_SetPrescaler ( TSI\_Type \* *base*, tsi\_electrode\_osc\_ -  
prescaler\_t *prescaler* ) [inline], [static]**

Parameters

|                  |                          |
|------------------|--------------------------|
| <i>base</i>      | TSI module base address. |
| <i>prescaler</i> | Prescaler value.         |

Returns

None.

**39.2.4.34 static void TSI\_HAL\_SetNumberOfScans ( TSI\_Type \* *base*,  
tsi\_n\_consecutive\_scans\_t *number* ) [inline], [static]**

## TSI HAL driver

### Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | TSI module base address. |
| <i>number</i> | Number of scans.         |

### Returns

None.

**39.2.4.35** `static void TSI_HAL_SetLowPowerScanInterval ( TSI_Type * base, tsi_low_power_interval_t interval ) [inline], [static]`

### Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>base</i>     | TSI module base address.     |
| <i>interval</i> | Interval for low power scan. |

### Returns

None.

**39.2.4.36** `static tsi_low_power_interval_t TSI_HAL_GetLowPowerScanInterval ( TSI_Type * base ) [inline], [static]`

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

Interval for low power scan.

**39.2.4.37** `static void TSI_HAL_SetLowPowerClock ( TSI_Type * base, uint32_t clock ) [inline], [static]`

Parameters

|              |                            |
|--------------|----------------------------|
| <i>base</i>  | TSI module base address.   |
| <i>clock</i> | Low power clock selection. |

**39.2.4.38** `static uint32_t TSI_HAL_GetLowPowerClock ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

Low power clock selection.

**39.2.4.39** `static void TSI_HAL_SetReferenceChargeCurrent ( TSI_Type * base, tsi_reference_osc_charge_current_t current ) [inline], [static]`

Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | TSI module base address. |
| <i>current</i> | The charge current.      |

Returns

None.

**39.2.4.40** `static tsi_reference_osc_charge_current_t TSI_HAL_GetReferenceChargeCurrent ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

The charge current.

**39.2.4.41** `static void TSI_HAL_SetElectrodeChargeCurrent ( TSI_Type * base, tsi_external_osc_charge_current_t current ) [inline], [static]`

## TSI HAL driver

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | TSI module base address. |
| <i>current</i> | Electrode current.       |

### Returns

None.

**39.2.4.42** `static tsi_external_osc_charge_current_t TSI_HAL_GetElectrodeChargeCurrent ( TSI_Type * base ) [inline], [static]`

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

Charge current.

**39.2.4.43** `static void TSI_HAL_SetScanModulo ( TSI_Type * base, uint32_t modulo ) [inline], [static]`

### Parameters

|               |                          |
|---------------|--------------------------|
| <i>base</i>   | TSI module base address. |
| <i>modulo</i> | Scan modulo value.       |

### Returns

None.

**39.2.4.44** `static uint32_t TSI_HAL_GetScanModulo ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

Scan modulo value.

**39.2.4.45** `static void TSI_HAL_SetActiveModeSource ( TSI_Type * base, uint32_t source ) [inline], [static]`

Parameters

|               |                                                          |
|---------------|----------------------------------------------------------|
| <i>base</i>   | TSI module base address.                                 |
| <i>source</i> | Active mode clock source (LPOSCCLK, MCGIRCLK, OSCERCLK). |

Returns

None.

**39.2.4.46** `static uint32_t TSI_HAL_GetActiveModeSource ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

Source value.

**39.2.4.47** `static void TSI_HAL_SetActiveModePrescaler ( TSI_Type * base, tsi_active_mode_prescaler_t prescaler ) [inline], [static]`

## TSI HAL driver

### Parameters

|                  |                          |
|------------------|--------------------------|
| <i>base</i>      | TSI module base address. |
| <i>prescaler</i> | Prescaler's value.       |

### Returns

None.

**39.2.4.48** `static uint32_t TSI_HAL_GetActiveModePrescaler ( TSI_Type * base )  
[inline], [static]`

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

Prescaler's value.

**39.2.4.49** `static void TSI_HAL_SetLowPowerChannel ( TSI_Type * base, uint32_t channel  
) [inline], [static]`

Only one channel can wake up MCU.

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | TSI module base address. |
| <i>channel</i> | Channel number.          |

### Returns

None.

**39.2.4.50** `static uint32_t TSI_HAL_GetLowPowerChannel ( TSI_Type * base )  
[inline], [static]`

Only one channel can wake up MCU.



## Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

## Returns

Channel number.

**39.2.4.51** `static void TSI_HAL_EnableChannel ( TSI_Type * base, uint32_t channel )  
[inline], [static]`

## Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | TSI module base address. |
| <i>channel</i> | Channel to be enabled.   |

## Returns

None.

**39.2.4.52** `static void TSI_HAL_EnableChannels ( TSI_Type * base, uint32_t  
channelsMask ) [inline], [static]`

The function enables channels by mask. It can set all at once.

## Parameters

|                     |                              |
|---------------------|------------------------------|
| <i>base</i>         | TSI module base address.     |
| <i>channelsMask</i> | Channels mask to be enabled. |

## Returns

None.

**39.2.4.53** `static void TSI_HAL_DisableChannel ( TSI_Type * base, uint32_t channel )  
[inline], [static]`

## TSI HAL driver

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | TSI module base address. |
| <i>channel</i> | Channel to be disabled.  |

### Returns

None.

**39.2.4.54** `static void TSI_HAL_DisableChannels ( TSI_Type * base, uint32_t channelsMask ) [inline], [static]`

The function disables channels by mask. It can set all at once.

### Parameters

|                     |                               |
|---------------------|-------------------------------|
| <i>base</i>         | TSI module base address.      |
| <i>channelsMask</i> | Channels mask to be disabled. |

### Returns

None.

**39.2.4.55** `static uint32_t TSI_HAL_GetEnabledChannel ( TSI_Type * base, uint32_t channel ) [inline], [static]`

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | TSI module base address. |
| <i>channel</i> | Channel to be checked.   |

### Returns

True - if channel is enabled, false - otherwise.

**39.2.4.56** `static uint32_t TSI_HAL_GetEnabledChannels ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

Channels mask that are enabled.

**39.2.4.57** `static uint16_t TSI_HAL_GetWakeUpChannelCounter ( TSI_Type * base )  
[inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

Wake up counter value.

**39.2.4.58** `static uint32_t TSI_HAL_GetCounter ( TSI_Type * base, uint32_t channel )  
[inline], [static]`

Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | TSI module base address. |
| <i>channel</i> | Index of TSI channel.    |

Returns

The counter value.

**39.2.4.59** `static void TSI_HAL_SetLowThreshold ( TSI_Type * base, uint32_t  
low_threshold ) [inline], [static]`

## TSI HAL driver

### Parameters

|                      |                          |
|----------------------|--------------------------|
| <i>base</i>          | TSI module base address. |
| <i>low_threshold</i> | Low counter threshold.   |

### Returns

None.

**39.2.4.60** `static void TSI_HAL_SetHighThreshold ( TSI_Type * base, uint32_t high_threshold ) [inline], [static]`

### Parameters

|                       |                          |
|-----------------------|--------------------------|
| <i>base</i>           | TSI module base address. |
| <i>high_threshold</i> | High counter threshold.  |

### Returns

None.

**39.2.4.61** `static uint32_t TSI_HAL_GetEnableStop ( TSI_Type * base ) [inline], [static]`

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

Number of scans.

**39.2.4.62** `static void TSI_HAL_EnableHardwareTriggerScan ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.63** `static void TSI_HAL_CurrentSourcePairSwapped ( TSI_Type * base )  
[inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.64** `static void TSI_HAL_CurrentSourcePairNotSwapped ( TSI_Type * base )  
[inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.65** `static uint32_t TSI_HAL_GetCurrentSourcePairSwapped ( TSI_Type * base )  
[inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

Current source pair swapped status.

---

## TSI HAL driver

**39.2.4.66** `static void TSI_HAL_SetMeasuredChannelNumber ( TSI_Type * base, uint32_t channel ) [inline], [static]`

Parameters

|                |                          |
|----------------|--------------------------|
| <i>base</i>    | TSI module base address. |
| <i>channel</i> | Channel number 0 ... 15. |

Returns

None.

**39.2.4.67** `static uint32_t TSI_HAL_GetMeasuredChannelNumber ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

uint32\_t Channel number 0 ... 15.

**39.2.4.68** `static void TSI_HAL_DmaTransferEnable ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

None.

**39.2.4.69** `static void TSI_HAL_DmaTransferDisable ( TSI_Type * base ) [inline], [static]`

## TSI HAL driver

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

None.

**39.2.4.70** `static uint32_t TSI_HAL_IsDmaTransferEnable ( TSI_Type * base ) [inline], [static]`

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

State of enable module flag.

**39.2.4.71** `static uint32_t TSI_HAL_GetCounter ( TSI_Type * base ) [inline], [static]`

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

### Returns

Accumulated scan counter value ticked by the reference clock.

**39.2.4.72** `static void TSI_HAL_SetMode ( TSI_Type * base, tsi_analog_mode_select_t mode ) [inline], [static]`

### Parameters

---



|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
| <i>mode</i> | Mode value.              |

Returns

None.

**39.2.4.73** `static tsi_analog_mode_select_t TSI_HAL_GetMode ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

`tsi_analog_mode_select_t` Mode value.

**39.2.4.74** `static uint32_t TSI_HAL_GetNoiseResult ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

`tsi_analog_mode_select_t` Mode value.

**39.2.4.75** `static void TSI_HAL_SetOscillatorVoltageRails ( TSI_Type * base, tsi_oscillator_voltage_rails_t dvolt ) [inline], [static]`

Parameters

---

## TSI HAL driver

|              |                          |
|--------------|--------------------------|
| <i>base</i>  | TSI module base address. |
| <i>dvolt</i> | The voltage rails.       |

Returns

None.

**39.2.4.76** `static tsi_oscilator_voltage_rails_t TSI_HAL_GetOscilatorVoltageRails ( TSI_Type * base ) [inline], [static]`

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | TSI module base address. |
|-------------|--------------------------|

Returns

*dvolt* The voltage rails..

### 39.3 TSI Peripheral Driver

#### 39.3.1 Overview

The section describes the programming interface of the TSI Peripheral driver

#### 39.3.2 Overview

The TSI peripheral driver measures the capacitance of electrodes (touch sensing) in various modes and provides API functions for input/output operations.

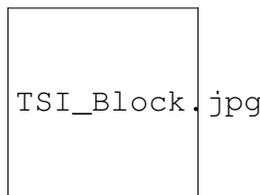


Figure 39.3.1: TSI Driver Block Diagram

#### 39.3.3 TSI Initialization

To initialize TSI, the application defines the startup configuration and handles to the initialization routine. Include the "fsl\_tsi\_driver.h" header file in source files to use TSI driver and operate touch sensing functionality.

This is an 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,
 .lpclks = kTsiLowPowerInterval_100ms,
 .amclks = kTsiActiveClkSource_BusClock,
 .ampsc = kTsiActiveModePrescaler_8div,
 .lpscnitv = kTsiLowPowerInterval_100ms,
 .thresh = 100,
 .thresl = 200,
};

// Set up the configuration of initialization structure
static const tsi_user_config_t myTsiConfig =
```

## TSI Peripheral Driver

```

{
 .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)
 {
 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;
}

```

## 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 \*usrData)  
*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

- TSI\_Type \*const [g\\_tsiBase](#) []  
*Table of base addresses for TSI instances.*
- const IRQn\_Type [g\\_tsiIrqId](#) [TSI\_INSTANCE\_COUNT]  
*Table to save TSI IRQ enumeration numbers defined in CMSIS header file.*

## TSI Peripheral Driver

- `tsi_state_t * g_tsiStatePtr` [TSI\_INSTANCE\_COUNT]  
*Table to save pointers to context data.*

### Initialization

- `tsi_status_t TSI_DRV_Init` (uint32\_t instance, `tsi_state_t *tsiState`, const `tsi_user_config_t *tsiUserConfig`)  
*Initializes a TSI instance for operation.*
- `tsi_status_t TSI_DRV_DeInit` (uint32\_t instance)  
*Shuts down the TSI by disabling interrupts and the peripheral.*
- `tsi_status_t TSI_DRV_EnableElectrode` (uint32\_t instance, const uint32\_t channel, const bool enable)  
*Enables/disables one electrode of the TSI module.*
- `uint32_t TSI_DRV_GetEnabledElectrodes` (uint32\_t instance)  
*Returns a mask of the enabled electrodes of the TSI module.*
- `tsi_status_t TSI_DRV_Measure` (uint32\_t instance)  
*Starts the measure cycle of the enabled electrodes.*
- `tsi_status_t TSI_DRV_MeasureBlocking` (uint32\_t instance)  
*Starts the measure cycle of the enabled electrodes in blocking mode.*
- `tsi_status_t TSI_DRV_AbortMeasure` (uint32\_t instance)  
*Aborts the measure cycle in non standard use of the driver.*
- `tsi_status_t TSI_DRV_GetCounter` (uint32\_t instance, const uint32\_t channel, uint16\_t \*counter)  
*Returns the last measured value.*
- `tsi_status_t TSI_DRV_GetStatus` (uint32\_t instance)  
*Returns the current status of the driver.*
- `tsi_status_t TSI_DRV_EnableLowPower` (uint32\_t instance)  
*Enters the low power mode of the TSI driver.*
- `tsi_status_t TSI_DRV_DisableLowPower` (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` (uint32\_t instance, uint32\_t \*lowestSignal)  
*Automatically recalibrates all important TSI settings.*
- `tsi_status_t TSI_DRV_SetCallbackFunc` (uint32\_t instance, const `tsi_callback_t pFuncCallback`, void \*usrData)  
*Sets the callback function that is called when the measure cycle ends.*
- `tsi_status_t TSI_DRV_ChangeMode` (uint32\_t instance, const `tsi_modes_t mode`)  
*Changes the current working operation mode.*
- `tsi_modes_t TSI_DRV_GetMode` (uint32\_t instance)  
*Returns the current working operation mode.*
- `tsi_status_t TSI_DRV_LoadConfiguration` (uint32\_t instance, const `tsi_modes_t mode`, const `tsi_operation_mode_t *operationMode`)  
*Loads the new configuration into a specific mode.*
- `tsi_status_t TSI_DRV_SaveConfiguration` (uint32\_t instance, const `tsi_modes_t mode`, `tsi_operation_mode_t *operationMode`)  
*Saves the TSI driver configuration for a specific mode.*

## 39.3.4 Data Structure Documentation

### 39.3.4.1 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 * usrData`

#### 39.3.4.1.0.68 Field Documentation

##### 39.3.4.1.0.68.1 tsi\_config\_t\* tsi\_user\_config\_t::config

A pointer to hardware configuration. Can't be NULL.

##### 39.3.4.1.0.68.2 tsi\_callback\_t tsi\_user\_config\_t::pCallBackFunc

A pointer to call back function of end of measurement.

##### 39.3.4.1.0.68.3 void\* tsi\_user\_config\_t::usrData

A user data of call back function.

### 39.3.4.2 struct tsi\_operation\_mode\_t

This is the 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](#)

#### 39.3.4.2.0.69 Field Documentation

##### 39.3.4.2.0.69.1 uint16\_t tsi\_operation\_mode\_t::enabledElectrodes

The back up of enabled electrodes for operation mode

##### 39.3.4.2.0.69.2 tsi\_config\_t tsi\_operation\_mode\_t::config

A hardware configuration.

## TSI Peripheral Driver

### 39.3.4.3 struct tsi\_state\_t

It must be created by the application code and the pointer is handled by the [TSI\\_DRV\\_Init](#) function to driver. The driver keeps all context 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]

#### 39.3.4.3.0.70 Field Documentation

##### 39.3.4.3.0.70.1 [tsi\\_status\\_t](#) tsi\_state\_t::status

Current status of the driver.

##### 39.3.4.3.0.70.2 [tsi\\_callback\\_t](#) tsi\_state\_t::pCallBackFunc

A pointer to call back function of end of measurement.

##### 39.3.4.3.0.70.3 void\* tsi\_state\_t::usrData

A user data pointer handled by call back function.

##### 39.3.4.3.0.70.4 [semaphore\\_t](#) tsi\_state\_t::irqSync

Used to wait for ISR to complete its measure business.

##### 39.3.4.3.0.70.5 [mutex\\_t](#) tsi\_state\_t::lock

Used by whole driver to secure the context data integrity.

##### 39.3.4.3.0.70.6 [mutex\\_t](#) tsi\_state\_t::lockChangeMode

Used by change mode function to secure the context data integrity.

##### 39.3.4.3.0.70.7 bool tsi\_state\_t::isBlockingMeasure

Used to internal indication of type of measurement.

##### 39.3.4.3.0.70.8 [tsi\\_modes\\_t](#) tsi\_state\_t::opMode

Storage of current operation mode.



### 39.3.4.3.0.70.9 `tsi_operation_mode_t tsi_state_t::opModesData[tsi_OpModeCnt]`

Data storage of individual operational modes.

### 39.3.4.3.0.70.10 `uint16_t tsi_state_t::counters[FSL_FEATURE_TSI_CHANNEL_COUNT]`

The mirror of last state of counter registers

## 39.3.5 Typedef Documentation

### 39.3.5.1 `typedef void(* tsi_callback_t)(uint32_t instance, void *usrData)`

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>usrData</i>  | - user data (type is void*), the user data are specified by function <a href="#">TSI_DRV_SetCall-BackFunc</a> |

Returns

- none

## 39.3.6 Enumeration Type Documentation

### 39.3.6.1 `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 the 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.

## TSI Peripheral Driver

### 39.3.7 Function Documentation

#### 39.3.7.1 `tsi_status_t TSI_DRV_Init ( uint32_t instance, tsi_state_t * tsiState, const tsi_user_config_t * tsiUserConfig )`

This function initializes the run-time state structure and prepares the entire peripheral to 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
}
```

#### 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 <code>tsi_user_config_t</code> . The user populates the members of this structure and passes the pointer of this structure into the function.                                                                                                             |

#### Returns

An error code or `kStatus_TSI_Success`.

#### 39.3.7.2 `tsi_status_t TSI_DRV_DeInit ( uint32_t instance )`

This function disables the TSI interrupts and the peripheral.

```
if(TSI_DRV_DeInit(0) != kStatus_TSI_Success)
{
 // Error, the TSI is not de-initialized
}
```

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>instance</i> | The TSI module instance. |
|-----------------|--------------------------|

Returns

An error code or kStatus\_TSI\_Success.

**39.3.7.3 tsi\_status\_t TSI\_DRV\_EnableElectrode ( 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.

**39.3.7.4 uint32\_t TSI\_DRV\_GetEnabledElectrodes ( uint32\_t instance )**

The function returns the mask of the enabled electrodes of the current mode.

```

uint32_t enabledElectrodeMask;
enabledElectrodeMask = TSI_DRV_GetEnabledElectrodes(0);

```

Parameters

---

## TSI Peripheral Driver

|                 |                          |
|-----------------|--------------------------|
| <i>instance</i> | The TSI module instance. |
|-----------------|--------------------------|

Returns

Mask of enabled electrodes for current mode.

### 39.3.7.5 `tsi_status_t TSI_DRV_Measure ( uint32_t instance )`

The function is non blocking. Therefore, the results can be obtained after the driver completes the measure cycle. The end of the measure cycle can be checked by pooling the [TSI\\_DRV\\_GetStatus](#) function or wait for registered callback function by using the [TSI\\_DRV\\_SetCallBackFunc](#) or [TSI\\_DRV\\_Init](#).

```
// Example of the 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
}
```

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>instance</i> | The TSI module instance. |
|-----------------|--------------------------|

Returns

An error code or `kStatus_TSI_Success`.

### 39.3.7.6 `tsi_status_t TSI_DRV_MeasureBlocking ( uint32_t instance )`

This function is blocking. Therefore, after the function call, the result of measured electrodes is available and can be obtained by calling the [TSI\\_DRV\\_GetCounter](#) function.

```
// Example of the TSI_DRV_Measure() function pooling style
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`.

### 39.3.7.7 `tsi_status_t TSI_DRV_AbortMeasure ( uint32_t instance )`

This function aborts the running measure cycle. It is designed for unexpected situations and 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`.

### 39.3.7.8 `tsi_status_t TSI_DRV_GetCounter ( uint32_t instance, const uint32_t channel, uint16_t * counter )`

This function returns the last measured value in the previous measure cycle. The data is buffered inside the 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 read
}
```

## TSI Peripheral Driver

### 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`.

### 39.3.7.9 `tsi_status_t TSI_DRV_GetStatus ( uint32_t instance )`

This function returns the current working status of the driver.

```
// Get the current status of TSI driver
tsi_status_t status;
status = TSI_DRV_GetStatus(0);
```

### Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>instance</i> | The TSI module instance. |
|-----------------|--------------------------|

### Returns

An current status of the driver.

### 39.3.7.10 `tsi_status_t TSI_DRV_EnableLowPower ( uint32_t instance )`

This function switches the driver to low power mode and immediately enables the low power functionality of the TSI peripheral. Before calling this function, the low power mode must be configured - Enable the right electrode and recalibrate the low power mode to get the best performance for this mode.

```
// Switch the driver to the low power mode
uint16_t signal;

// The first time is needed to 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`.

#### 39.3.7.11 `tsi_status_t TSI_DRV_DisableLowPower ( 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`.

#### 39.3.7.12 `tsi_status_t TSI_DRV_Recalibrate ( uint32_t instance, uint32_t * lowestSignal )`

This function forces the driver to start the recalibration procedure for the current operation mode to get the best possible TSI hardware settings. The computed setting is stored into the operation mode data and can be loaded and saved by the [TSI\\_DRV\\_LoadConfiguration](#) or the [TSI\\_DRV\\_SaveConfiguration](#) functions.

### Warning

The function could take more time to return 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
}
```

## TSI Peripheral Driver

### 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`.

#### 39.3.7.13 `tsi_status_t TSI_DRV_SetCallbackFunc ( uint32_t instance, const tsi_callback_t pFuncCallback, void * usrData )`

This function sets up or clears, (parameter `pFuncCallback = NULL`), the callback function pointer which is called after each measure cycle ends. The user can also set the custom user data, that is handled by the parameter to a call back function. One function can be called by more sources.

```
// 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 moment
}

// 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 moment
}
```

### Parameters

|                      |                                               |
|----------------------|-----------------------------------------------|
| <i>instance</i>      | The TSI module instance.                      |
| <i>pFuncCallback</i> | The pointer to application call back function |
| <i>usrData</i>       | The user data pointer                         |

### Returns

An error code or `kStatus_TSI_Success`.

#### 39.3.7.14 `tsi_status_t TSI_DRV_ChangeMode ( uint32_t instance, const tsi_modes_t mode )`

This function changes the working operation mode of the 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.

### 39.3.7.15 tsi\_modes\_t TSI\_DRV\_GetMode ( uint32\_t instance )

This function returns the current working operation mode of the 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.

### 39.3.7.16 tsi\_status\_t TSI\_DRV\_LoadConfiguration ( uint32\_t instance, const tsi\_modes\_t mode, const tsi\_operation\_mode\_t \* operationMode )

This function loads the new configuration into a specific mode. This can be used when the calibrated data are stored in any NVM to load after startup of the MCU to avoid run recalibration that takes more 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
}
```

## TSI Peripheral Driver

### 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`.

### 39.3.7.17 `tsi_status_t TSI_DRV_SaveConfiguration ( uint32_t instance, const tsi_modes_t mode, tsi_operation_mode_t * operationMode )`

This function saves the configuration of a specific mode. This can be used when the calibrated data should be stored in any backup memory to load after the start of the MCU to avoid running the recalibration that takes more 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`.

## 39.3.8 Variable Documentation

### 39.3.8.1 `TSI_Type* const g_tsiBase[]`

### 39.3.8.2 `const IRQn_Type g_tsilrqld[TSI_INSTANCE_COUNT]`

### 39.3.8.3 `tsi_state_t* g_tsiStatePtr[TSI_INSTANCE_COUNT]`

---

## Chapter 40

# Universal Asynchronous Receiver/Transmitter (UART)

### 40.1 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](#)

## UART HAL driver

### 40.2 UART HAL driver

#### 40.2.1 Overview

#### 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 }
- enum `uart_parity_mode_t` {  
`kUartParityDisabled` = 0x0U,  
`kUartParityEven` = 0x2U,  
`kUartParityOdd` = 0x3U }
- enum `uart_bit_count_per_char_t` {  
`kUart8BitsPerChar` = 0U,  
`kUart9BitsPerChar` = 1U }
- enum `uart_operation_config_t` {  
`kUartOperates` = 0U,  
`kUartStops` = 1U }
- enum `uart_receiver_source_t` {  
`kUartLoopBack` = 0U,  
`kUartSingleWire` = 1U }
- enum `uart_wakeup_method_t` {  
`kUartIdleLineWake` = 0U,  
`kUartAddrMarkWake` = 1U }
- enum `uart_idle_line_select_t` {  
`kUartIdleLineAfterStartBit` = 0U,  
`kUartIdleLineAfterStopBit` = 1U }
- enum `uart_break_char_length_t` {  
`kUartBreakChar10BitMinimum` = 0U,  
`kUartBreakChar13BitMinimum` = 1U }
- 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_iso7816_transfer_protocoltype_t {`
  - `kUartIso7816TransfertType0 = 0U,`
  - `kUartIso7816TransfertType1 = 1U }`
  - UART ISO7816 transport protocol type options.*
- `enum uart_iso7816_onack_config_t {`
  - `kUartIso7816OnackEnable = 0U,`
  - `kUartIso7816OnackDisable = 1U }`
  - UART ISO7816 ONACK generation.*
- `enum uart_iso7816_anack_config_t {`
  - `kUartIso7816AnackDisable = 0U,`
  - `kUartIso7816AnackEnable = 1U }`
  - UART ISO7816 ANACK generation.*
- `enum uart_iso7816_initd_config_t {`
  - `kUartIso7816InitdDisable = 0U,`
  - `kUartIso7816InitdEnable = 1U }`
  - UART ISO7816 Initial Character detection.*
- `enum uart_status_flag_t {`
  - `kUartTxDataRegEmpty = 0U << UART_SHIFT | UART_S1_TDRE_SHIFT,`
  - `kUartTxComplete = 0U << UART_SHIFT | UART_S1_TC_SHIFT,`
  - `kUartRxDataRegFull = 0U << UART_SHIFT | UART_S1_RDRF_SHIFT,`
  - `kUartIdleLineDetect = 0U << UART_SHIFT | UART_S1_IDLE_SHIFT,`
  - `kUartRxOverrun = 0U << UART_SHIFT | UART_S1_OR_SHIFT,`
  - `kUartNoiseDetect = 0U << UART_SHIFT | UART_S1_NF_SHIFT,`
  - `kUartFrameErr = 0U << UART_SHIFT | UART_S1_FE_SHIFT,`
  - `kUartParityErr = 0U << UART_SHIFT | UART_S1_PF_SHIFT,`
  - `kUartRxActiveEdgeDetect = 1U << UART_SHIFT | UART_S2_RXEDGIF_SHIFT,`
  - `kUartRxActive = 1U << UART_SHIFT | UART_S2_RAF_SHIFT }`
  - UART status flags.*
- `enum uart_interrupt_t {`
  - `kUartIntRxActiveEdge = 0U << UART_SHIFT | UART_BDH_RXEDGIE_SHIFT,`
  - `kUartIntTxDataRegEmpty = 1U << UART_SHIFT | UART_C2_TIE_SHIFT,`
  - `kUartIntTxComplete = 1U << UART_SHIFT | UART_C2_TCIE_SHIFT,`
  - `kUartIntRxDataRegFull = 1U << UART_SHIFT | UART_C2_RIE_SHIFT,`
  - `kUartIntIdleLine = 1U << UART_SHIFT | UART_C2_ILIE_SHIFT,`
  - `kUartIntRxOverrun = 2U << UART_SHIFT | UART_C3_ORIE_SHIFT,`
  - `kUartIntNoiseErrFlag = 2U << UART_SHIFT | UART_C3_NEIE_SHIFT,`
  - `kUartIntFrameErrFlag = 2U << UART_SHIFT | UART_C3_FEIE_SHIFT,`
  - `kUartIntParityErrFlag = 2U << UART_SHIFT | UART_C3_PEIE_SHIFT }`

## UART HAL driver

- UART interrupt configuration structure, default settings are 0 (disabled).*

  - enum `uart_iso7816_interrupt_t` {
    - `kUartIntIso7816RxThresholdExceeded` = 0U,
    - `kUartIntIso7816TxThresholdExceeded` = 1U,
    - `kUartIntIso7816GuardTimerViolated` = 2U,
    - `kUartIntIso7816AtrDurationTimer` = 3U,
    - `kUartIntIso7816InitialCharDetected` = 4U,
    - `kUartIntIso7816BlockWaitTimer` = 5U,
    - `kUartIntIso7816CharWaitTimer` = 6U,
    - `kUartIntIso7816WaitTimer` = 7U }

*UART ISO7816 specific interrupt configuration.*

## UART Common Configurations

- void `UART_HAL_Init` (UART\_Type \*base)
  - Initializes the UART controller.*
- static void `UART_HAL_EnableTransmitter` (UART\_Type \*base)
  - Enables the UART transmitter.*
- static void `UART_HAL_DisableTransmitter` (UART\_Type \*base)
  - Disables the UART transmitter.*
- static bool `UART_HAL_IsTransmitterEnabled` (UART\_Type \*base)
  - Gets the UART transmitter enabled/disabled configuration setting.*
- static void `UART_HAL_EnableReceiver` (UART\_Type \*base)
  - Enables the UART receiver.*
- static void `UART_HAL_DisableReceiver` (UART\_Type \*base)
  - Disables the UART receiver.*
- static bool `UART_HAL_IsReceiverEnabled` (UART\_Type \*base)
  - Gets the UART receiver enabled/disabled configuration setting.*
- `uart_status_t` `UART_HAL_SetBaudRate` (UART\_Type \*base, uint32\_t sourceClockInHz, uint32\_t baudRate)
  - Configures the UART baud rate.*
- void `UART_HAL_SetBaudRateDivisor` (UART\_Type \*base, uint16\_t baudRateDivisor)
  - Sets the UART baud rate modulo divisor value.*
- static void `UART_HAL_SetBitCountPerChar` (UART\_Type \*base, `uart_bit_count_per_char_t` bitCountPerChar)
  - Configures the number of bits per character in the UART controller.*
- void `UART_HAL_SetParityMode` (UART\_Type \*base, `uart_parity_mode_t` parityMode)
  - Configures the parity mode in the UART controller.*
- static uint32\_t `UART_HAL_GetDataRegAddr` (UART\_Type \*base)
  - Get UART tx/rx data register address.*

## UART Interrupts and DMA

- void `UART_HAL_SetIntMode` (UART\_Type \*base, `uart_interrupt_t` interrupt, bool enable)
  - Configures the UART module interrupts to enable/disable various interrupt sources.*
- bool `UART_HAL_GetIntMode` (UART\_Type \*base, `uart_interrupt_t` interrupt)
  - Returns whether the UART module interrupts is enabled/disabled.*

## UART Transfer Functions

- void [UART\\_HAL\\_Putchar](#) (UART\_Type \*base, uint8\_t data)  
*This function allows the user to send an 8-bit character from the UART data register.*
- void [UART\\_HAL\\_Putchar9](#) (UART\_Type \*base, uint16\_t data)  
*This function allows the user to send a 9-bit character from the UART data register.*
- void [UART\\_HAL\\_Getchar](#) (UART\_Type \*base, uint8\_t \*readData)  
*This function gets a received 8-bit character from the UART data register.*
- void [UART\\_HAL\\_Getchar9](#) (UART\_Type \*base, uint16\_t \*readData)  
*This function gets a received 9-bit character from the UART data register.*
- void [UART\\_HAL\\_SendDataPolling](#) (UART\_Type \*base, const uint8\_t \*txBuff, uint32\_t txSize)  
*Send out multiple bytes of data using polling method.*
- [uart\\_status\\_t](#) [UART\\_HAL\\_ReceiveDataPolling](#) (UART\_Type \*base, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Receive multiple bytes of data using polling method.*

## UART Status Flags

- bool [UART\\_HAL\\_GetStatusFlag](#) (UART\_Type \*base, [uart\\_status\\_flag\\_t](#) statusFlag)  
*Gets all UART status flag states.*
- [uart\\_status\\_t](#) [UART\\_HAL\\_ClearStatusFlag](#) (UART\_Type \*base, [uart\\_status\\_flag\\_t](#) statusFlag)  
*Clears an individual and specific UART status flag.*

## UART Special Feature Configurations

- static void [UART\\_HAL\\_SetLoopCmd](#) (UART\_Type \*base, bool enable)  
*Configures the UART loopback operation.*
- static void [UART\\_HAL\\_SetReceiverSource](#) (UART\_Type \*base, [uart\\_receiver\\_source\\_t](#) source)  
*Configures the UART single-wire operation.*
- static void [UART\\_HAL\\_SetTransmitterDir](#) (UART\_Type \*base, [uart\\_singlewire\\_txdirection\\_t](#) direction)  
*Configures the UART transmit direction while in single-wire mode.*
- [uart\\_status\\_t](#) [UART\\_HAL\\_PutReceiverInStandbyMode](#) (UART\_Type \*base)  
*Places the UART receiver in standby mode.*
- static void [UART\\_HAL\\_PutReceiverInNormalMode](#) (UART\_Type \*base)  
*Places the UART receiver in normal mode (disable standby mode operation).*
- static bool [UART\\_HAL\\_IsReceiverInStandby](#) (UART\_Type \*base)  
*Determines if the UART receiver is currently in standby mode.*
- static void [UART\\_HAL\\_SetReceiverWakeupMethod](#) (UART\_Type \*base, [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](#) (UART\_Type \*base)  
*Gets the UART receiver wakeup method (idle-line or address-mark) from standby mode.*
- void [UART\\_HAL\\_ConfigIdleLineDetect](#) (UART\_Type \*base, uint8\_t idleLine, uint8\_t rxWakeIdleDetect)  
*Configures the operation options of the UART idle line detect.*
- static void [UART\\_HAL\\_SetBreakCharTransmitLength](#) (UART\_Type \*base, [uart\\_break\\_char\\_length\\_t](#) length)  
*Configures the UART break character transmit length.*

## UART HAL driver

- static void `UART_HAL_SetBreakCharCmd` (UART\_Type \*base, bool enable)  
*Configures the UART transmit send break character operation.*
- void `UART_HAL_SetMatchAddress` (UART\_Type \*base, bool matchAddrMode1, bool matchAddrMode2, uint8\_t matchAddrValue1, uint8\_t matchAddrValue2)  
*Configures the UART match address mode control operation.*

## 40.2.2 Enumeration Type Documentation

### 40.2.2.1 enum uart\_status\_t

### 40.2.2.2 enum uart\_stop\_bit\_count\_t

These constants define the number of allowable stop bits to configure in a UART base.

Enumerator

*kUartOneStopBit* one stop bit  
*kUartTwoStopBit* two stop bits

### 40.2.2.3 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

### 40.2.2.4 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 base supports the desired number of data bits per UART character.

Enumerator

*kUart8BitsPerChar* 8-bit data characters  
*kUart9BitsPerChar* 9-bit data characters



#### 40.2.2.5 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.

#### 40.2.2.6 enum uart\_receiver\_source\_t

Enumerator

*kUartLoopBack* Internal loop back mode.  
*kUartSingleWire* Single wire mode.

#### 40.2.2.7 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.

#### 40.2.2.8 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.

#### 40.2.2.9 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 base.

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)

## UART HAL driver

***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)

### 40.2.2.10 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.

Enumerator

***kUartSinglewireTxdirIn*** UART Single-Wire mode TXDIR input.

***kUartSinglewireTxdirOut*** UART Single-Wire mode TXDIR output.

### 40.2.2.11 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

### 40.2.2.12 enum uart\_iso7816\_transfer\_protocoltype\_t

This provides constants for the UART ISO7816 transport protocol types.

Enumerator

***kUartIso7816TransfertType0*** Transfer type 0.

***kUartIso7816TransfertType1*** Transfer type 1.

### 40.2.2.13 enum uart\_iso7816\_onack\_config\_t

This provides constants for the UART ISO7816 module ONACK generation.

Enumerator

***kUartIso7816OnackEnable*** Enable ONACK generation.

***kUartIso7816OnackDisable*** Disable ONACK generation.

#### 40.2.2.14 enum uart\_iso7816\_anack\_config\_t

This provides constants for the UART ISO7816 module ANACK generation.

Enumerator

- kUartIso7816AnackDisable* Disable ANACK generation.
- kUartIso7816AnackEnable* Enable ANACK generation.

#### 40.2.2.15 enum uart\_iso7816\_initd\_config\_t

This provides constants for the UART ISO7816 module Initial generation.

Enumerator

- kUartIso7816InitdDisable* Disable Initial Character detection.
- kUartIso7816InitdEnable* Enable Initial Character detection.

#### 40.2.2.16 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* Rx Overrun, sets when new data is received before data is read from receive register.
- kUartNoiseDetect* Rx 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.

#### 40.2.2.17 enum uart\_interrupt\_t

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

Enumerator

- kUartIntRxActiveEdge* RX Active Edge.

## UART HAL driver

*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.

### 40.2.2.18 enum uart\_iso7816\_interrupt\_t

This enum contains the settings for all of the UART ISO7816 feature specific interrupt configurations.

Enumerator

*kUartIntIso7816RxThreasholdExceeded* Receive Threshold Exceeded.  
*kUartIntIso7816TxThresholdExceeded* TransmitThresholdExceeded.  
*kUartIntIso7816GuardTimerViolated* Guard Timer Violated.  
*kUartIntIso7816AtrDurationTimer* ATR Duration Timer.  
*kUartIntIso7816InitialCharDetected* Initial Character Detected.  
*kUartIntIso7816BlockWaitTimer* Block Wait Timer.  
*kUartIntIso7816CharWaitTimer* Character Wait Timer.  
*kUartIntIso7816WaitTimer* Wait Timer.

## 40.2.3 Function Documentation

### 40.2.3.1 void UART\_HAL\_Init ( UART\_Type \* *base* )

This function initializes the module to a known state.

Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | UART module base pointer. |
|-------------|---------------------------|

### 40.2.3.2 static void UART\_HAL\_EnableTransmitter ( UART\_Type \* *base* ) [inline], [static]

This function allows the user to enable the UART transmitter.

Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | UART module base pointer. |
|-------------|---------------------------|

#### 40.2.3.3 **static void UART\_HAL\_DisableTransmitter ( UART\_Type \* *base* ) [inline], [static]**

This function allows the user to disable the UART transmitter.

Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | UART module base pointer. |
|-------------|---------------------------|

#### 40.2.3.4 **static bool UART\_HAL\_IsTransmitterEnabled ( UART\_Type \* *base* ) [inline], [static]**

This function allows the user to get the setting of the UART transmitter.

Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | UART module base pointer. |
|-------------|---------------------------|

Returns

The state of UART transmitter enable(true)/disable(false) setting.

#### 40.2.3.5 **static void UART\_HAL\_EnableReceiver ( UART\_Type \* *base* ) [inline], [static]**

This function allows the user to enable the UART receiver.

Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | UART module base pointer. |
|-------------|---------------------------|

#### 40.2.3.6 **static void UART\_HAL\_DisableReceiver ( UART\_Type \* *base* ) [inline], [static]**

This function allows the user to disable the UART receiver.

## UART HAL driver

### Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | UART module base pointer. |
|-------------|---------------------------|

### 40.2.3.7 static bool UART\_HAL\_IsReceiverEnabled ( UART\_Type \* *base* ) [inline], [static]

This function allows the user to get the setting of the UART receiver.

### Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | UART module base pointer. |
|-------------|---------------------------|

### Returns

The state of UART receiver enable(true)/disable(false) setting.

### 40.2.3.8 uart\_status\_t UART\_HAL\_SetBaudRate ( UART\_Type \* *base*, 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 bases 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>base</i>            | UART module base pointer.      |
| <i>sourceClockInHz</i> | UART source input clock in Hz. |
| <i>baudRate</i>        | UART desired baud rate.        |

### Returns

An error code or kStatus\_UART\_Success

### 40.2.3.9 void UART\_HAL\_SetBaudRateDivisor ( UART\_Type \* *base*, 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>base</i>             | UART module base pointer.                  |
| <i>baudRate-Divisor</i> | The baud rate modulo division "SBR" value. |

**40.2.3.10 static void UART\_HAL\_SetBitCountPerChar ( UART\_Type \* *base*,  
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`.

## Parameters

|                        |                                                                    |
|------------------------|--------------------------------------------------------------------|
| <i>base</i>            | UART module base pointer.                                          |
| <i>bitCountPerChar</i> | Number of bits per char (8, 9, or 10, depending on the UART base). |

**40.2.3.11 void UART\_HAL\_SetParityMode ( UART\_Type \* *base*, 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>base</i>       | UART module base pointer.                                                                  |
| <i>parityMode</i> | Parity mode setting (enabled, disable, odd, even - see <code>parity_mode_t</code> struct). |

**40.2.3.12 static uint32\_t UART\_HAL\_GetDataRegAddr ( UART\_Type \* *base* )  
[inline], [static]**

## Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | UART module base pointer. |
|-------------|---------------------------|

## Returns

UART tx/rx data register address.

---

## UART HAL driver

**40.2.3.13** void UART\_HAL\_SetIntMode ( UART\_Type \* *base*, uart\_interrupt\_t *interrupt*, bool *enable* )



## Parameters

|                  |                                    |
|------------------|------------------------------------|
| <i>base</i>      | UART module base pointer.          |
| <i>interrupt</i> | UART interrupt configuration data. |
| <i>enable</i>    | true: enable, false: disable.      |

**40.2.3.14 bool UART\_HAL\_GetIntMode ( UART\_Type \* *base*, uart\_interrupt\_t *interrupt* )**

## Parameters

|                  |                                    |
|------------------|------------------------------------|
| <i>base</i>      | UART module base pointer.          |
| <i>interrupt</i> | UART interrupt configuration data. |

## Returns

true: enable, false: disable.

**40.2.3.15 void UART\_HAL\_Putchar ( UART\_Type \* *base*, uint8\_t *data* )**

## Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | UART module base pointer.       |
| <i>data</i> | The data to send of size 8-bit. |

**40.2.3.16 void UART\_HAL\_Putchar9 ( UART\_Type \* *base*, uint16\_t *data* )**

## Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | UART module base pointer.       |
| <i>data</i> | The data to send of size 9-bit. |

**40.2.3.17 void UART\_HAL\_Getchar ( UART\_Type \* *base*, uint8\_t \* *readData* )**

## UART HAL driver

Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>base</i>     | UART module base pointer.                                |
| <i>readData</i> | The received data read from data register of size 8-bit. |

### 40.2.3.18 void UART\_HAL\_Getchar9 ( UART\_Type \* *base*, uint16\_t \* *readData* )

Parameters

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <i>base</i>     | UART module base pointer.                                |
| <i>readData</i> | The received data read from data register of size 9-bit. |

### 40.2.3.19 void UART\_HAL\_SendDataPolling ( UART\_Type \* *base*, const uint8\_t \* *txBuff*, uint32\_t *txSize* )

This function only supports 8-bit transaction.

Parameters

|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>base</i>   | UART module base pointer.                           |
| <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.            |

### 40.2.3.20 uart\_status\_t UART\_HAL\_ReceiveDataPolling ( UART\_Type \* *base*, uint8\_t \* *rxBuff*, uint32\_t *rxSize* )

This function only supports 8-bit transaction.

Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | UART module base pointer.                               |
| <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.

40.2.3.21 `bool UART_HAL_GetStatusFlag ( UART_Type * base, uart_status_flag_t statusFlag )`

## UART HAL driver

### Parameters

|                   |                           |
|-------------------|---------------------------|
| <i>base</i>       | UART module base pointer. |
| <i>statusFlag</i> | Status flag name.         |

### Returns

Whether the current status flag is set(true) or not(false).

#### 40.2.3.22 `uart_status_t` **UART\_HAL\_ClearStatusFlag** ( `UART_Type * base`, `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>base</i>       | UART module base pointer.              |
| <i>statusFlag</i> | The desired UART status flag to clear. |

### Returns

An error code or `kStatus_UART_Success`.

#### 40.2.3.23 `static void` **UART\_HAL\_SetLoopCmd** ( `UART_Type * base`, `bool enable` ) `[inline]`, `[static]`

This function enables or disables the UART loopback operation.

### Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>base</i>   | UART module base pointer.                                                        |
| <i>enable</i> | The UART loopback mode configuration, either disabled (false) or enabled (true). |

#### 40.2.3.24 `static void` **UART\_HAL\_SetReceiverSource** ( `UART_Type * base`, `uart_receiver_source_t source` ) `[inline]`, `[static]`

This function enables or disables the UART single-wire operation. In some UART bases 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>base</i>   | UART module base pointer.                |
| <i>source</i> | The UART single-wire mode configuration. |

#### 40.2.3.25 **static void UART\_HAL\_SetTransmitterDir ( UART\_Type \* *base*, uart\_singlewire\_txdir\_t *direction* ) [inline], [static]**

This function configures the transmitter direction when the UART is configured for single-wire operation.

Parameters

|                  |                                                                                                                                                                                                |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | UART module base pointer.                                                                                                                                                                      |
| <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> ). |

#### 40.2.3.26 **uart\_status\_t UART\_HAL\_PutReceiverInStandbyMode ( UART\_Type \* *base* )**

This function, when called, places the UART receiver into standby mode. In some UART bases, 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>base</i> | UART module base pointer. |
|-------------|---------------------------|

Returns

Error code or `kStatus_UART_Success`.

#### 40.2.3.27 **static void UART\_HAL\_PutReceiverInNormalMode ( UART\_Type \* *base* ) [inline], [static]**

This function, when called, places the UART receiver into normal mode and out of standby mode.

## UART HAL driver

### Parameters

|             |                           |
|-------------|---------------------------|
| <i>base</i> | UART module base pointer. |
|-------------|---------------------------|

#### 40.2.3.28 **static bool UART\_HAL\_IsReceiverInStandby ( UART\_Type \* *base* ) [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>base</i> | UART module base pointer. |
|-------------|---------------------------|

### Returns

The UART receiver is in normal mode (false) or standby mode (true).

#### 40.2.3.29 **static void UART\_HAL\_SetReceiverWakeupMethod ( UART\_Type \* *base*, 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>base</i>   | UART module base pointer.                                                                                             |
| <i>method</i> | The UART receiver wakeup method options: kUartIdleLineWake - Idle-line wake or kUartAddrMarkWake - address-mark wake. |

#### 40.2.3.30 **static uart\_wakeup\_method\_t UART\_HAL\_GetReceiverWakeupMethod ( UART\_Type \* *base* ) [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>base</i> | UART module base pointer. |
|-------------|---------------------------|

**Returns**

The UART receiver wakeup from standby method, false: kUartIdleLineWake (idle-line wake) or true: kUartAddrMarkWake (address-mark wake).

**40.2.3.31 void UART\_HAL\_ConfigIdleLineDetect ( UART\_Type \* *base*, uint8\_t *idleLine*, uint8\_t *rxWakeIdleDetect* )**

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`.

**Parameters**

|                          |                                                                                                                                                                                                                                                   |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | UART module base pointer.                                                                                                                                                                                                                         |
| <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 |

**40.2.3.32 static void UART\_HAL\_SetBreakCharTransmitLength ( UART\_Type \* *base*, 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 bases 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>base</i>   | UART module base pointer.                                                                                                                        |
| <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. |

**40.2.3.33 static void UART\_HAL\_SetBreakCharCmd ( UART\_Type \* *base*, 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.

## UART HAL driver

### Parameters

|               |                                                                                                                                                                                         |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | UART module base pointer.                                                                                                                                                               |
| <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. |

**40.2.3.34 void UART\_HAL\_SetMatchAddress ( UART\_Type \* *base*, bool *matchAddrMode1*, bool *matchAddrMode2*, uint8\_t *matchAddrValue1*, uint8\_t *matchAddrValue2* )**

(Note: Feature available on select UART bases)

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>base</i>             | UART module base pointer.                                                 |
| <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.              |



## 40.3 UART Peripheral driver

### 40.3.1 Overview

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.

### 40.3.2 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 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.

### 40.3.3 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, and the number of bits per data word.

### 40.3.4 UART Initialization

1. To initialize the UART driver, call the `UART_DRV_Init()` function and pass the instance number of the UART peripheral, 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 Peripheral driver

```
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);
```

### 40.3.5 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\(\)](#) functions.

The blocking (async) transmit and receive functions include the [UART\\_DRV\\_SendDataBlocking\(\)](#) and [UART\\_DRV\\_ReceiveDataBlocking\(\)](#) functions.

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(uartInstance, sourceBuff, byteCount, 1); // function
 won't return until transmit is complete
UART_DRV_ReceiveDataBlocking(uartInstance, 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 txBytesRemaining, rxBytesRemaining;

// assume pTxBuff and txSize have been initialized
UART_DRV_SendData(uartInstance, pTxBuff, txSize);

// 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(uartInstance, &txBytesRemaining) ==
 kStatus_UART_TxBusy);

// for receive, assume rxBuff is set up to receive data and rxSize is initialized
UART_DRV_ReceiveData(uartInstance, rxBuff, rxSize);

// 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(uartInstance, &rxBytesRemaining) ==
 kStatus_UART_RxBusy);
```

### Files

- file [fsl\\_uart\\_driver.h](#)  
Some devices count the UART instances with LPUART(e.g., KL27) or UART0(e.g., KL25) together.

### 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...](#)

### Typedefs

- typedef void(\* [uart\\_rx\\_callback\\_t](#))(uint32\_t instance, void \*uartState)  
UART receive callback function type.
- typedef void(\* [uart\\_tx\\_callback\\_t](#))(uint32\_t instance, void \*uartState)  
UART transmit callback function type.

### Variables

- UART\_Type \*const [g\\_uartBase](#) [UART\_INSTANCE\_COUNT]  
Table of base addresses for UART instances.
- const IRQn\_Type [g\\_uartRxTxIrqId](#) [UART\_INSTANCE\_COUNT]  
Table to save UART IRQ enumeration numbers defined in the CMSIS header file.

### 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.
- [uart\\_status\\_t](#) [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)

## UART Peripheral driver

- Installs the callback function for the UART receive.*

  - `uart_tx_callback_t UART_DRV_InstallTxCallback` (uint32\_t instance, `uart_tx_callback_t` function, uint8\_t \*txBuff, void \*callbackParam)
- Installs the callback function for the UART transmit.*

  - `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.*

### 40.3.6 Data Structure Documentation

#### 40.3.6.1 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.*
- `uart_tx_callback_t txCallback`  
*Callback to invoke after transmitting byte.*
- void \* `txCallbackParam`  
*Transmit callback parameter pointer.*

## UART Peripheral driver

### 40.3.6.1.0.71 Field Documentation

40.3.6.1.0.71.1 `uint8_t uart_state_t::txFifoEntryCount`

40.3.6.1.0.71.2 `const uint8_t* uart_state_t::txBuff`

40.3.6.1.0.71.3 `uint8_t* uart_state_t::rxBuff`

40.3.6.1.0.71.4 `volatile size_t uart_state_t::txSize`

40.3.6.1.0.71.5 `volatile size_t uart_state_t::rxSize`

40.3.6.1.0.71.6 `volatile bool uart_state_t::isTxBusy`

40.3.6.1.0.71.7 `volatile bool uart_state_t::isRxBusy`

40.3.6.1.0.71.8 `volatile bool uart_state_t::isTxBlocking`

40.3.6.1.0.71.9 `volatile bool uart_state_t::isRxBlocking`

40.3.6.1.0.71.10 `semaphore_t uart_state_t::txIrqSync`

40.3.6.1.0.71.11 `semaphore_t uart_state_t::rxIrqSync`

40.3.6.1.0.71.12 `uart_rx_callback_t uart_state_t::rxCallback`

40.3.6.1.0.71.13 `void* uart_state_t::rxCallbackParam`

40.3.6.1.0.71.14 `uart_tx_callback_t uart_state_t::txCallback`

40.3.6.1.0.71.15 `void* uart_state_t::txCallbackParam`

### 40.3.6.2 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)*

## 40.3.7 Function Documentation

### 40.3.7.1 `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, un-gates 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 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`.

### 40.3.7.2 `uart_status_t` UART\_DRV\_Deinit ( `uint32_t` *instance* )

This function disables the UART interrupts, the transmitter and receiver, and flushes the FIFOs (for modules that support FIFOs).

#### Parameters

---

## UART Peripheral driver

|                 |                           |
|-----------------|---------------------------|
| <i>instance</i> | The UART instance number. |
|-----------------|---------------------------|

Returns

An error code or kStatus\_UART\_Success.

### 40.3.7.3 `uart_rx_callback_t` `UART_DRV_InstallRxCallback ( uint32_t instance, uart_rx_callback_t function, uint8_t * rxBuff, void * callbackParam, bool alwaysEnableRxIrq )`

Note

After a callback is installed, it bypasses part of the UART IRQHandler 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 receive IRQ or not.                                                             |

Returns

Former UART receive callback function pointer.

### 40.3.7.4 `uart_tx_callback_t` `UART_DRV_InstallTxCallback ( uint32_t instance, uart_tx_callback_t function, uint8_t * txBuff, void * callbackParam )`

Note

After a callback is installed, it bypasses part of the UART IRQHandler logic. Therefore, the callback needs to handle the txBuff and txSize indexes.



## Parameters

|                      |                                                                                                        |
|----------------------|--------------------------------------------------------------------------------------------------------|
| <i>instance</i>      | The UART instance number.                                                                              |
| <i>function</i>      | The UART transmit callback function.                                                                   |
| <i>txBuff</i>        | The transmit buffer used inside IRQHandler. This buffer must be kept as long as the callback is alive. |
| <i>callbackParam</i> | The UART transmit callback parameter pointer.                                                          |

## Returns

Former UART transmit callback function pointer.

#### 40.3.7.5 **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.

#### 40.3.7.6 **uart\_status\_t UART\_DRV\_SendData ( uint32\_t *instance*, const uint8\_t \* *txBuff*, uint32\_t *txSize* )**

A non-blocking (also known as asynchronous) 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).

## UART Peripheral driver

### 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`.

### 40.3.7.7 `uart_status_t UART_DRV_GetTransmitStatus ( uint32_t instance, uint32_t * bytesRemaining )`

When performing an a-sync 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

|                        |                                                                                                            |
|------------------------|------------------------------------------------------------------------------------------------------------|
| <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. |

### 40.3.7.8 `uart_status_t UART_DRV_AbortSendingData ( uint32_t instance )`

During an a-sync 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. |

**40.3.7.9 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.

**40.3.7.10 uart\_status\_t UART\_DRV\_ReceiveData ( uint32\_t instance, uint8\_t \* rxBuff, uint32\_t rxSize )**

A non-blocking (also known as asynchronous) 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).

## UART Peripheral driver

### 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`.

#### 40.3.7.11 `uart_status_t UART_DRV_GetReceiveStatus ( uint32_t instance, uint32_t * bytesRemaining )`

When performing an a-sync 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.

### Parameters

|                        |                                                                                                                         |
|------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <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 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. |

#### 40.3.7.12 `uart_status_t UART_DRV_AbortReceivingData ( uint32_t instance )`

During an a-sync 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_No-TransmitInProgress</i> | No receive is currently in progress. |

### 40.3.8 Variable Documentation

#### 40.3.8.1 `UART_Type* const g_uartBase[UART_INSTANCE_COUNT]`

---

## UART Peripheral driver

## Chapter 41 Reference (VREF)

### 41.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Voltage Reference (VREF) of Kinetis devices.

#### Modules

- [VREF HAL Driver](#)
- [VREF Peripheral driver](#)

## VREF HAL Driver

### 41.2 VREF HAL Driver

#### 41.2.1 Overview

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

#### Files

- file [fsl\\_vref\\_hal.h](#)

#### Data Structures

- struct [vref\\_user\\_config\\_t](#)  
*The description structure for the VREF module. [More...](#)*

#### Enumerations

- enum [vref\\_status\\_t](#) {  
[kStatus\\_VREF\\_Success](#) = 0x0U,  
[kStatus\\_VREF\\_InvalidArgument](#) = 0x1U,  
[kStatus\\_VREF\\_Failed](#) = 0x2U }  
*Definitions*
- enum [vref\\_buffer\\_mode\\_t](#) {  
[kVrefModeBandgapOnly](#) = 0x0U,  
[kVrefModeTightRegulationBuffer](#) = 0x2U }  
*VREF modes.*

#### VREF related feature APIs

##### API

- void [VREF\\_HAL\\_Init](#) (VREF\_Type \*base)  
*Initialize VREF module to default state.*
- void [VREF\\_HAL\\_Configure](#) (VREF\_Type \*base, const [vref\\_user\\_config\\_t](#) \*userConfigPtr)  
*Configure VREF module to known state.*
- static void [VREF\\_HAL\\_Enable](#) (VREF\_Type \*base)  
*Enable VREF module.*
- static void [VREF\\_HAL\\_Disable](#) (VREF\_Type \*base)  
*Disable VREF module.*
- static void [VREF\\_HAL\\_SetInternalRegulatorCmd](#) (VREF\_Type \*base, bool enable)  
*Set VREF internal regulator to Enable.*
- static void [VREF\\_HAL\\_SetTrimVal](#) (VREF\_Type \*base, uint8\_t trimValue)  
*Set trim value for voltage reference.*
- static uint8\_t [VREF\\_HAL\\_GetTrimVal](#) (VREF\_Type \*base)  
*Read value of trim meaning output voltage.*



- static void **VREF\_HAL\_WaitVoltageStable** (VREF\_Type \*base)  
*Wait to internal voltage stable.*
- static void **VREF\_HAL\_SetBufferMode** (VREF\_Type \*base, **vref\_buffer\_mode\_t** mode)  
*Set buffer mode.*

## 41.2.2 Data Structure Documentation

### 41.2.2.1 struct vref\_user\_config\_t

#### Data Fields

- uint8\_t **trimValue**  
*Trim bits.*
- bool **regulatorEnable**  
*Enable regulator.*
- **vref\_buffer\_mode\_t** **bufferMode**  
*Buffer mode selection.*

## 41.2.3 Enumeration Type Documentation

### 41.2.3.1 enum vref\_status\_t

VREF status return codes

Enumerator

- kStatus\_VREF\_Success** Success.
- kStatus\_VREF\_InvalidArgument** Invalid argument existed.
- kStatus\_VREF\_Failed** Execution failed.

### 41.2.3.2 enum vref\_buffer\_mode\_t

Enumerator

- kVrefModeBandgapOnly** Bandgap on only. For stabilization and startup
- kVrefModeTightRegulationBuffer** Tight regulation buffer enabled.

## 41.2.4 Function Documentation

### 41.2.4.1 void VREF\_HAL\_Init ( VREF\_Type \* base )

## VREF HAL Driver

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | VREF module base number. |
|-------------|--------------------------|

**41.2.4.2 void VREF\_HAL\_Configure ( VREF\_Type \* *base*, const vref\_user\_config\_t \* *userConfigPtr* )**

Parameters

|                      |                                                                        |
|----------------------|------------------------------------------------------------------------|
| <i>base</i>          | VREF module base number.                                               |
| <i>userConfigPtr</i> | Pointer to the initialization structure. See the "vref_user_config_t". |

**41.2.4.3 static void VREF\_HAL\_Enable ( VREF\_Type \* *base* ) [inline], [static]**

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | VREF module base number. |
|-------------|--------------------------|

**41.2.4.4 static void VREF\_HAL\_Disable ( VREF\_Type \* *base* ) [inline], [static]**

Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | VREF module base number. |
|-------------|--------------------------|

**41.2.4.5 static void VREF\_HAL\_SetInternalRegulatorCmd ( VREF\_Type \* *base*, bool *enable* ) [inline], [static]**

Cannot be enabled in very low-power modes!

Parameters

|               |                                                                                                                                                                          |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | VREF module base number.                                                                                                                                                 |
| <i>enable</i> | Enables or disables internal regulator <ul style="list-style-type: none"> <li>• true : Internal regulator enable</li> <li>• false: Internal regulator disable</li> </ul> |

41.2.4.6 static void VREF\_HAL\_SetTrimVal ( VREF\_Type \* *base*, uint8\_t *trimValue* )  
[inline], [static]

## VREF HAL Driver

### Parameters

|                  |                                                                            |
|------------------|----------------------------------------------------------------------------|
| <i>base</i>      | VREF module base number.                                                   |
| <i>trimValue</i> | Value of trim register to set output reference voltage (max 0x3F (6-bit)). |

**41.2.4.7** `static uint8_t VREF_HAL_GetTrimVal ( VREF_Type * base ) [inline], [static]`

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | VREF module base number. |
|-------------|--------------------------|

### Returns

Six-bit value of trim setting.

**41.2.4.8** `static void VREF_HAL_WaitVoltageStable ( VREF_Type * base ) [inline], [static]`

### Parameters

|             |                          |
|-------------|--------------------------|
| <i>base</i> | VREF module base number. |
|-------------|--------------------------|

**41.2.4.9** `static void VREF_HAL_SetBufferMode ( VREF_Type * base, vref_buffer_mode_t mode ) [inline], [static]`

### Parameters

|             |                                                                                                                                                                                                                                                                                                                                    |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | VREF module base number.                                                                                                                                                                                                                                                                                                           |
| <i>mode</i> | Defines mode to be set. <ul style="list-style-type: none"> <li>• kVrefModeBandgapOnly : Set Bandgap on only</li> <li>• kVrefModeHighPowerBuffer : Set High power buffer mode</li> <li>• kVrefModeLowPowerBuffer : Set Low power buffer mode</li> <li>• kVrefModeTightRegulationBuffer: Set Tight regulation buffer mode</li> </ul> |

## 41.3 VREF Peripheral driver

### 41.3.1 Overview

This section describes the programming interface of the VREF Peripheral driver. The VREF Peripheral driver configures the Voltage Reference (VREF). It handles initialization and configuration of Voltage Reference (VREF) module.

### 41.3.2 VREF Initialization

To initialize the VREF module, call the `VREF_DRV_Init()` function and pass in the user configuration structure. This function automatically enables the WDOG module and clock.

This example code shows how to initialize and configure the driver:

```

vref_user_config_t vrefConfig =
{
#if FSL_FEATURE_VREF_HAS_CHOP_OSC
 .chopOscEnable = true,
#endif
 .trimValue = 0,
 .regulatorEnable = true,
#if FSL_FEATURE_VREF_HAS_COMPENSATION
 .soccEnable = true,
#endif
 .bufferMode = kVrefModeHighPowerBuffer
};

// Initialize VREF
VREF_DRV_Init(VREF_IDX, &vrefConfig);

```

### VREF set output result

To set the resulting VREF by approximately  $\pm 0.5$  mV for each step, call `VREF_DRV_SetTrimValue()` function and pass the TRIM value.

```

// These bits change the resulting VREF
VREF_DRV_SetTrimValue(VREF_IDX, trimValue);

```

### Files

- file `fsl_vref_driver.h`

### Functions

- `vref_status_t VREF_DRV_Init` (uint32\_t instance, const `vref_user_config_t *userConfigPtr`)  
*Initializes the VREF module.*
- `vref_status_t VREF_DRV_Deinit` (uint32\_t instance)

## VREF Peripheral driver

- De-initializes the VREF module.*

  - [vref\\_status\\_t VREF\\_DRV\\_SetTrimValue](#) (uint32\_t instance, uint8\_t trimValue)  
*Sets the TRIM bits value.*
  - static [uint8\\_t VREF\\_DRV\\_GetTrimValue](#) (uint32\_t instance)  
*Get TRIM bits value was set.*
  - static [vref\\_status\\_t VREF\\_DRV\\_SetRegulator](#) (uint32\_t instance, bool enable)  
*Enables or disables the regulator.*
  - static [vref\\_status\\_t VREF\\_DRV\\_SetBufferMode](#) (uint32\_t instance, [vref\\_buffer\\_mode\\_t](#) bufferMode)  
*Sets the buffer mode.*

### 41.3.3 Function Documentation

#### 41.3.3.1 [vref\\_status\\_t VREF\\_DRV\\_Init](#) ( uint32\_t instance, const [vref\\_user\\_config\\_t](#) \* userConfigPtr )

Parameters

|                      |                                                                                          |
|----------------------|------------------------------------------------------------------------------------------|
| <i>instance</i>      | VREF instance.                                                                           |
| <i>userConfigPtr</i> | Pointer to the initialization structure. See the " <a href="#">vref_user_config_t</a> ". |

Returns

Execution status.

#### 41.3.3.2 [vref\\_status\\_t VREF\\_DRV\\_Deinit](#) ( uint32\_t instance )

Parameters

|                 |                |
|-----------------|----------------|
| <i>instance</i> | VREF instance. |
|-----------------|----------------|

Returns

Execution status.

#### 41.3.3.3 [vref\\_status\\_t VREF\\_DRV\\_SetTrimValue](#) ( uint32\_t instance, uint8\_t trimValue )

These bits change the resulting VREF by approximately +/- 0.5 mV for each step. For minimum and maximum voltage reference output values, see the Data Sheet for this chip.

## Parameters

|                  |                  |
|------------------|------------------|
| <i>instance</i>  | VREF instance.   |
| <i>trimValue</i> | TRIM bits value. |

## Returns

Execution status.

**41.3.3.4 static uint8\_t VREF\_DRV\_GetTrimValue ( uint32\_t *instance* ) [inline], [static]**

## Parameters

|                 |                |
|-----------------|----------------|
| <i>instance</i> | VREF instance. |
|-----------------|----------------|

## Returns

Actual TRIM bits value.

**41.3.3.5 static vref\_status\_t VREF\_DRV\_SetRegulator ( uint32\_t *instance*, bool *enable* ) [inline], [static]**

This bit is used to enable the internal 1.75 V regulator to produce a constant internal voltage supply to reduce the sensitivity to the external supply noise and variation. To keep the regulator enabled in very low power modes, see the Chip Configuration details for a description. This bit is set during factory trimming of the VREF voltage and should be written to 1 to achieve the performance stated in the data sheet.

## Parameters

|                 |                                                                                                                                                                          |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i> | VREF instance.                                                                                                                                                           |
| <i>enable</i>   | Enables or disables internal regulator <ul style="list-style-type: none"> <li>• true : Internal regulator enable</li> <li>• false: Internal regulator disable</li> </ul> |

## Returns

Execution status.

## VREF Peripheral driver

**41.3.3.6 static vref\_status\_t VREF\_DRV\_SetBufferMode ( uint32\_t *instance*, vref\_buffer\_mode\_t *bufferMode* ) [inline], [static]**

These bits select the buffer modes for the Voltage Reference module.

- Buffer mode = 0x00: The internal VREF bandgap is enabled to generate an accurate 1.2 V output that can be trimmed with the TRM register TRIM[5:0] bit field. The bandgap requires some time for startup and stabilization. SC[VREFST] can be monitored to determine if the stabilization and startup is complete.
- Buffer mode = 0x01: The internal VREF bandgap is on. The high power buffer is enabled to generate a buffered 1.2 V voltage to VREF\_OUT.
- Buffer mode = 0x02: The internal VREF bandgap is on. The high power buffer is enabled to generate a buffered 1.2 V voltage to VREF\_OUT.

VREF\_OUT generated by high and low buffer modes can also be used as a reference to internal analog peripherals such as an ADC channel or analog comparator input. If those modes is entered from the standby mode, there is a delay before the buffer output is settled at the final value. A 100 nF capacitor is required to connect between the VREF\_OUT pin and VSSA.

Parameters

|                   |                    |
|-------------------|--------------------|
| <i>instance</i>   | VREF instance.     |
| <i>bufferMode</i> | Buffer mode value. |

Returns

Execution status.



---

## Chapter 42

### Watchdog Timer (WDOG)

#### 42.1 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

### 42.2 WDOG HAL driver

#### 42.2.1 Overview

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

#### Data Structures

- struct [wdog\\_work\\_mode\\_t](#)  
*Describes wdog work mode structure. [More...](#)*
- struct [wdog\\_config\\_t](#)  
*Describes wdog configuration structure. [More...](#)*

#### Enumerations

- enum [wdog\\_clk\\_src\\_t](#) {  
    [kWdogLpoClkSrc](#) = 0U,  
    [kWdogAlternateClkSrc](#) = 1U }  
*Describes wdog clock source structure.*
- enum [wdog\\_clk\\_prescaler\\_t](#) {  
    [kWdogClkPrescalerDivide1](#) = 0x0U,  
    [kWdogClkPrescalerDivide2](#) = 0x1U,  
    [kWdogClkPrescalerDivide3](#) = 0x2U,  
    [kWdogClkPrescalerDivide4](#) = 0x3U,  
    [kWdogClkPrescalerDivide5](#) = 0x4U,  
    [kWdogClkPrescalerDivide6](#) = 0x5U,  
    [kWdogClkPrescalerDivide7](#) = 0x6U,  
    [kWdogClkPrescalerDivide8](#) = 0x7U }  
*Describes the selection of the clock prescaler.*
- enum [wdog\\_status\\_t](#) {  
    [kStatus\\_WDOG\\_Success](#) = 0x0U,  
    [kStatus\\_WDOG\\_Fail](#) = 0x1U,  
    [kStatus\\_WDOG\\_NotInitialized](#) = 0x2U,  
    [kStatus\\_WDOG\\_NullArgument](#) = 0x3U }  
*wdog status return codes.*

#### Watchdog HAL.

- static void [WDOG\\_HAL\\_Enable](#) (WDOG\_Type \*base)  
*Enables the Watchdog module.*
- static void [WDOG\\_HAL\\_Disable](#) (WDOG\_Type \*base)  
*Disables the Watchdog module.*
- static bool [WDOG\\_HAL\\_IsEnable](#) (WDOG\_Type \*base)  
*Checks whether the WDOG is enabled.*
- void [WDOG\\_HAL\\_SetConfig](#) (WDOG\_Type \*base, const [wdog\\_config\\_t](#) \*configPtr)

- *Sets the WDOG common configure.*
- static void [WDOG\\_HAL\\_SetIntCmd](#) (WDOG\_Type \*base, bool enable)  
*Enables and disables the Watchdog interrupt.*
- static bool [WDOG\\_HAL\\_GetIntFlag](#) (WDOG\_Type \*base)  
*Gets the Watchdog interrupt status.*
- static void [WDOG\\_HAL\\_ClearIntStatusFlag](#) (WDOG\_Type \*base)  
*Clears the Watchdog interrupt flag.*
- static void [WDOG\\_HAL\\_SetTimeoutValue](#) (WDOG\_Type \*base, uint32\_t timeoutCount)  
*Set the Watchdog timeout value.*
- static uint32\_t [WDOG\\_HAL\\_GetTimerOutputValue](#) (WDOG\_Type \*base)  
*Gets the Watchdog timer output.*
- static void [WDOG\\_HAL\\_SetWindowValue](#) (WDOG\_Type \*base, uint32\_t windowValue)  
*Sets the Watchdog window value.*
- static void [WDOG\\_HAL\\_Unlock](#) (WDOG\_Type \*base)  
*Unlocks the Watchdog register written.*
- static void [WDOG\\_HAL\\_Refresh](#) (WDOG\_Type \*base)  
*Refreshes the Watchdog timer.*
- static void [WDOG\\_HAL\\_ResetSystem](#) (WDOG\_Type \*base)  
*Resets the chip using the Watchdog.*
- void [WDOG\\_HAL\\_Init](#) (WDOG\_Type \*base)  
*Restores the WDOG module to reset value.*

## 42.2.2 Data Structure Documentation

### 42.2.2.1 struct wdog\_work\_mode\_t

#### Data Fields

- bool [kWdogEnableInWaitMode](#)  
*Enables or disables wdog in wait mode.*
- bool [kWdogEnableInStopMode](#)  
*Enables or disables wdog in stop mode.*
- bool [kWdogEnableInDebugMode](#)  
*Enables or disables wdog in debug mode.*

### 42.2.2.2 struct wdog\_config\_t

#### Data Fields

- bool [wdogEnable](#)  
*Enables or disables wdog.*
- [wdog\\_clk\\_src\\_t](#) clkSrc  
*Clock source select.*
- [wdog\\_clk\\_prescaler\\_t](#) prescaler  
*Clock prescaler value.*
- [wdog\\_work\\_mode\\_t](#) workMode  
*Configures wdog work mode in debug stop and wait mode.*
- bool [updateEnable](#)

## WDOG HAL driver

- *Update write-once register enable.*  
 • bool `intEnable`  
*Enables or disables wdog interrupt.*
- bool `winEnable`  
*Enables or disables wdog window mode.*
- uint32\_t `windowValue`  
*Window value.*
- uint32\_t `timeoutValue`  
*Timeout value.*

### 42.2.3 Enumeration Type Documentation

#### 42.2.3.1 enum wdog\_clk\_src\_t

Enumerator

- kWdogLpoClkSrc* wdog clock sourced from LPO
- kWdogAlternateClkSrc* wdog clock sourced from alternate clock source

#### 42.2.3.2 enum wdog\_clk\_prescaler\_t

Enumerator

- kWdogClkPrescalerDivide1* Divided by 1.
- kWdogClkPrescalerDivide2* Divided by 2.
- kWdogClkPrescalerDivide3* Divided by 3.
- kWdogClkPrescalerDivide4* Divided by 4.
- kWdogClkPrescalerDivide5* Divided by 5.
- kWdogClkPrescalerDivide6* Divided by 6.
- kWdogClkPrescalerDivide7* Divided by 7.
- kWdogClkPrescalerDivide8* Divided by 8.

#### 42.2.3.3 enum wdog\_status\_t

Enumerator

- kStatus\_WDOG\_Success* WDOG operation Succeed.
- kStatus\_WDOG\_Fail* WDOG operation Failed.
- kStatus\_WDOG\_NotInitialized* WDOG is not initialized yet.
- kStatus\_WDOG\_NullArgument* Argument is NULL.

## 42.2.4 Function Documentation

### 42.2.4.1 `static void WDOG_HAL_Enable ( WDOG_Type * base ) [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>base</i> | The WDOG peripheral base address |
|-------------|----------------------------------|

### 42.2.4.2 `static void WDOG_HAL_Disable ( WDOG_Type * base ) [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>base</i> | The WDOG peripheral base address |
|-------------|----------------------------------|

### 42.2.4.3 `static bool WDOG_HAL_IsEnable ( WDOG_Type * base ) [inline], [static]`

This function checks whether the WDOG is enabled.

Parameters

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

Returns

false means WDOG is disabled, true means WDOG is enabled.

### 42.2.4.4 `void WDOG_HAL_SetConfig ( WDOG_Type * base, const wdog_config_t * configPtr )`

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 HAL driver

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>base</i>      | The WDOG peripheral base address |
| <i>configPtr</i> | The common configure of the WDOG |

### 42.2.4.5 static void WDOG\_HAL\_SetIntCmd ( WDOG\_Type \* *base*, 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.

Parameters

|               |                                                                                  |
|---------------|----------------------------------------------------------------------------------|
| <i>base</i>   | The WDOG peripheral base address                                                 |
| <i>enable</i> | false means disable watchdog interrupt and true means enable watchdog interrupt. |

### 42.2.4.6 static bool WDOG\_HAL\_GetIntFlag ( WDOG\_Type \* *base* ) [inline], [static]

This function gets the WDOG interrupt flag.

Parameters

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

Returns

Watchdog interrupt status, false means interrupt not asserted, true means interrupt asserted.

### 42.2.4.7 static void WDOG\_HAL\_ClearIntStatusFlag ( WDOG\_Type \* *base* ) [inline], [static]

This function clears the WDOG interrupt flag.

## Parameters

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

**42.2.4.8 static void WDOG\_HAL\_SetTimeoutValue ( WDOG\_Type \* *base*, uint32\_t *timeoutCount* ) [inline], [static]**

This function sets the WDOG\_TOVAL value. It should be ensured that the time-out value for the Watchdog 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>base</i>         | The WDOG peripheral base address                      |
| <i>timeoutCount</i> | watchdog timeout value, count of watchdog clock tick. |

**42.2.4.9 static uint32\_t WDOG\_HAL\_GetTimerOutputValue ( WDOG\_Type \* *base* ) [inline], [static]**

This function gets the WDOG\_TMROUT value.

## Parameters

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

## Returns

Current value of watchdog timer counter.

**42.2.4.10 static void WDOG\_HAL\_SetWindowValue ( WDOG\_Type \* *base*, uint32\_t *windowValue* ) [inline], [static]**

This function sets the WDOG\_WIN value. 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.

## WDOG HAL driver

### Parameters

|                    |                                  |
|--------------------|----------------------------------|
| <i>base</i>        | The WDOG peripheral base address |
| <i>windowValue</i> | watchdog window value.           |

#### 42.2.4.11 **static void WDOG\_HAL\_Unlock ( WDOG\_Type \* *base* ) [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>base</i> | The WDOG peripheral base address |
|-------------|----------------------------------|

#### 42.2.4.12 **static void WDOG\_HAL\_Refresh ( WDOG\_Type \* *base* ) [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>base</i> | The WDOG peripheral base address |
|-------------|----------------------------------|

#### 42.2.4.13 **static void WDOG\_HAL\_ResetSystem ( WDOG\_Type \* *base* ) [inline], [static]**

This function resets the chip using WDOG.

### Parameters

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

#### 42.2.4.14 **void WDOG\_HAL\_Init ( WDOG\_Type \* *base* )**

This function restores the WDOG module to reset value.



## Parameters

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

## WDOG Peripheral driver

### 42.3 WDOG Peripheral driver

#### 42.3.1 Overview

This section describes the programming interface of the WDOG Peripheral driver. The WDOG driver configures and initializes the WDOG.

#### 42.3.2 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 = kWdogClockPrescalerValueDevidel, // 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);
```

#### 42.3.3 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".

#### 42.3.4 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.

#### 42.3.5 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.

### 42.3.6 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 //
}
```

## Watchdog Driver

Data structure for Watchdog initialization

This structure is used when initializing the WDOG during the `wdog_init` function call. It contains all WDOG configurations.

- `wdog_status_t WDOG_DRV_Init` (const `wdog_config_t *userConfigPtr`)  
*Initializes the Watchdog.*
- `wdog_status_t WDOG_DRV_Deinit` (void)  
*Shuts down the Watchdog.*
- void `WDOG_DRV_Refresh` (void)  
*Refreshes the Watchdog.*
- bool `WDOG_DRV_IsRunning` (void)  
*Gets the Watchdog running status.*
- void `WDOG_DRV_ResetSystem` (void)  
*Resets the MCU by the Watchdog.*

### 42.3.7 Function Documentation

#### 42.3.7.1 `wdog_status_t WDOG_DRV_Init ( const wdog_config_t * userConfigPtr )`

This function initializes the WDOG. When called, the WDOG runs according to the requirements of the configuration.

Parameters

---

## WDOG Peripheral driver

|                      |                                                                  |
|----------------------|------------------------------------------------------------------|
| <i>userConfigPtr</i> | Watchdog user configure data structure, see #wdog_user_config_t. |
|----------------------|------------------------------------------------------------------|

### 42.3.7.2 wdog\_status\_t WDOG\_DRV\_Deinit ( void )

This function shuts down the WDOG.

### 42.3.7.3 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.

### 42.3.7.4 bool WDOG\_DRV\_IsRunning ( void )

This function gets the WDOG running status.

Returns

watchdog running status, false means not running, true means running

### 42.3.7.5 void WDOG\_DRV\_ResetSystem ( void )

This function resets the MCU by using the WDOG.

---

## Chapter 43

# Inter-Peripheral Crossbar Switch (XBAR)

### 43.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the Inter-Peripheral Crossbar Switch (XBAR) block of Kinetis devices.

#### Modules

- [XBAR HAL driver](#)
- [XBAR Peripheral driver](#)

## XBAR HAL driver

### 43.2 XBAR HAL driver

#### 43.2.1 Overview

This section describes the programming interface of the XBAR HAL driver. This layer contains APIs for XBARA (XBAR in KM3x device case) and XBARB (if present) module.

#### Enumerations

- enum `xbar_active_edge_t` {  
`kXbarEdgeNone` = 0U,  
`kXbarEdgeRising` = 1U,  
`kXbarEdgeFalling` = 2U,  
`kXbarEdgeRisingAndFalling` = 3U }  
*XBAR active edge for detection.*
- enum `xbar_status_t` {  
`kStatus_XBAR_Success` = 0U,  
`kStatus_XBAR_InvalidArgument` = 1U,  
`kStatus_XBAR_Initialized` = 2U,  
`kStatus_XBAR_Failed` = 3U }  
*Defines XBAR status return codes.*

#### Functions

- void `XBARA_HAL_Init` (`XBARA_Type *baseAddr`)  
*Initializes the XBARA module to the reset state.*
- static void `XBARA_HAL_SetOutSel` (`XBARA_Type *baseAddr`, `uint32_t outIndex`, `uint32_t input`)  
*Selects which of the shared inputs `XBARA_IN[*]` is muxed to selected output `XBARA_OUT[*]`.*
- static `uint32_t XBARA_HAL_GetOutSel` (`XBARA_Type *baseAddr`, `uint32_t outIndex`)  
*Gets input `XBARA_IN[*]` muxed to selected output `XBARA_OUT[*]`.*
- static void `XBARA_HAL_SetDMAOutCmd` (`XBARA_Type *baseAddr`, `uint32_t outIndex`, `bool enable`)  
*Sets the DMA function on the corresponding `XBARA_OUT[*]` output.*
- static void `XBARA_HAL_SetIntOutCmd` (`XBARA_Type *baseAddr`, `uint32_t outIndex`, `bool enable`)  
*Sets the interrupt function on the corresponding `XBARA_OUT[*]` output.*
- static `bool XBARA_HAL_GetDMAOutCmd` (`XBARA_Type *baseAddr`, `uint32_t outIndex`)  
*Checks whether the DMA function is enabled or disabled on the corresponding `XBARA_OUT[*]` output.*
- static `bool XBARA_HAL_GetIntOutCmd` (`XBARA_Type *baseAddr`, `uint32_t outIndex`)  
*Checks whether the interrupt function is enabled or disabled on the corresponding `XBARA_OUT[*]` output.*
- static void `XBARA_HAL_SetOutActiveEdge` (`XBARA_Type *baseAddr`, `uint32_t outIndex`, `xbar_active_edge_t edge`)  
*Selects which edges on the corresponding `XBARA_OUT[*]` output cause STSn to assert.*

- static `xbar_active_edge_t XBARA_HAL_GetOutActiveEdge` (`XBARA_Type *baseAddr`, `uint32_t outIndex`)  
*Gets which edges on the corresponding XBARA\_OUT[\*] output cause STSn to assert.*
- static void `XBARA_HAL_ClearEdgeDetectionStatus` (`XBARA_Type *baseAddr`, `uint32_t outIndex`)  
*Clears the edge detection status for the corresponding XBARA\_OUT[\*] output.*
- static bool `XBARA_HAL_GetEdgeDetectionStatus` (`XBARA_Type *baseAddr`, `uint32_t outIndex`)  
*Gets the edge detection status for the corresponding XBARA\_OUT[\*] output.*
- void `XBARB_HAL_Init` (`XBARB_Type *baseAddr`)  
*Initializes the XBARB module to the reset state.*
- static void `XBARB_HAL_SetOutSel` (`XBARB_Type *baseAddr`, `uint32_t outIndex`, `uint32_t input`)  
*Selects which of the shared inputs XBARB\_IN[\*] is muxed to selected output XBARB\_OUT[\*].*
- static `uint32_t XBARB_HAL_GetOutSel` (`XBARB_Type *baseAddr`, `uint32_t outIndex`)  
*Gets input XBARB\_IN[\*] muxed to selected output XBARB\_OUT[\*].*

## 43.2.2 Enumeration Type Documentation

### 43.2.2.1 enum xbar\_active\_edge\_t

Enumerator

- kXbarEdgeNone*** Edge detection status bit never asserts.
- kXbarEdgeRising*** Edge detection status bit asserts on rising edges.
- kXbarEdgeFalling*** Edge detection status bit asserts on falling edges.
- kXbarEdgeRisingAndFalling*** Edge detection status bit asserts on rising and falling edges.

### 43.2.2.2 enum xbar\_status\_t

Enumerator

- kStatus\_XBAR\_Success*** Success.
- kStatus\_XBAR\_InvalidArgument*** Invalid argument existed.
- kStatus\_XBAR\_Initialized*** Xbar has been already initialized.
- kStatus\_XBAR\_Failed*** Execution failed.

## 43.2.3 Function Documentation

### 43.2.3.1 void XBARA\_HAL\_Init ( XBARA\_Type \* baseAddr )

## XBAR HAL driver

### Parameters

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

#### 43.2.3.2 **static void XBARA\_HAL\_SetOutSel ( XBARA\_Type \* *baseAddr*, uint32\_t *outIndex*, uint32\_t *input* ) [inline], [static]**

This function selects which of the shared inputs XBARA\_IN[\*] is muxed to selected output XBARA\_OUT[\*].

### Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>baseAddr</i> | Register base address for XBARA module.            |
| <i>input</i>    | Input to be muxed to selected XBARA_OUT[*] output. |
| <i>outIndex</i> | Selected output XBARA_OUT[*].                      |

#### 43.2.3.3 **static uint32\_t XBARA\_HAL\_GetOutSel ( XBARA\_Type \* *baseAddr*, uint32\_t *outIndex* ) [inline], [static]**

This function gets input XBARA\_IN[\*] muxed to selected output XBARA\_OUT[\*].

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Register base address for XBAR module. |
| <i>outIndex</i> | Selected XBARA_OUT[*] output.          |

### Returns

Input XBARA\_IN[\*] muxed to selected XBARA\_OUT[\*] output.

#### 43.2.3.4 **static void XBARA\_HAL\_SetDMAOutCmd ( XBARA\_Type \* *baseAddr*, uint32\_t *outIndex*, bool *enable* ) [inline], [static]**

This function sets the DMA function on the corresponding XBARA\_OUT[\*]. When the interrupt is enabled, the output INT\_REQn reflects the value STSn. When the interrupt is disabled, INT\_REQn remains low. The interrupt request is cleared by writing a 1 to STSn.



## Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>baseAddr</i> | Register base address for XBARA module.       |
| <i>outIndex</i> | Selected XBARA_OUT[*] output.                 |
| <i>enable</i>   | Bool value for enable or disable DMA request. |

**43.2.3.5 static void XBARA\_HAL\_SetIntOutCmd ( XBARA\_Type \* *baseAddr*, uint32\_t *outIndex*, bool *enable* ) [inline], [static]**

This function sets the interrupt function on the corresponding XBARA\_OUT[\*]. When enabled, DMA\_REQn presents the value STSn. When disabled, the DMA\_REQn output remains low.

## Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>baseAddr</i> | Register base address for XBARA module.     |
| <i>outIndex</i> | Selected XBARA_OUT[*] output.               |
| <i>enable</i>   | Bool value for enable or disable interrupt. |

**43.2.3.6 static bool XBARA\_HAL\_GetDMAOutCmd ( XBARA\_Type \* *baseAddr*, uint32\_t *outIndex* ) [inline], [static]**

## Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>baseAddr</i> | Register base address for XBARA module. |
| <i>outIndex</i> | Selected XBARA_OUT[*] output.           |

## Returns

DMA function is enabled (true) or disabled (false).

**43.2.3.7 static bool XBARA\_HAL\_GetIntOutCmd ( XBARA\_Type \* *baseAddr*, uint32\_t *outIndex* ) [inline], [static]**

## Parameters

---

## XBAR HAL driver

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>baseAddr</i> | Register base address for XBARA module. |
| <i>outIndex</i> | Selected XBARA_OUT[*] output.           |

Returns

Interrupt function is enabled (true) or disabled (false).

### 43.2.3.8 **static void XBARA\_HAL\_SetOutActiveEdge ( XBARA\_Type \* *baseAddr*, uint32\_t *outIndex*, xbar\_active\_edge\_t *edge* ) [inline], [static]**

This function selects which edges on the corresponding XBARA\_OUT[\*] output cause STSn to assert.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>baseAddr</i> | Register base address for XBAR module. |
| <i>outIndex</i> | Selected XBARA_OUT[*] output.          |
| <i>edge</i>     | Active edge for edge detection.        |

### 43.2.3.9 **static xbar\_active\_edge\_t XBARA\_HAL\_GetOutActiveEdge ( XBARA\_Type \* *baseAddr*, uint32\_t *outIndex* ) [inline], [static]**

This function gets which edges on the corresponding XBARA\_OUT[\*] output cause STSn to assert.

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>baseAddr</i> | Register base address for XBARA module. |
| <i>outIndex</i> | Selected XBARA_OUT[*] output.           |

Returns

Active edge for edge detection on corresponding XBARA\_OUT[\*] output.

### 43.2.3.10 **static void XBARA\_HAL\_ClearEdgeDetectionStatus ( XBARA\_Type \* *baseAddr*, uint32\_t *outIndex* ) [inline], [static]**

## Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>baseAddr</i> | Register base address for XBARA module. |
| <i>outIndex</i> | Selected XBARA_OUT[*] output.           |

**43.2.3.11 static bool XBARA\_HAL\_GetEdgeDetectionStatus ( XBARA\_Type \* *baseAddr*, uint32\_t *outIndex* ) [inline], [static]**

## Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>baseAddr</i> | Register base address for XBARA module. |
| <i>outIndex</i> | Selected XBARA_OUT[*] output.           |

## Returns

Active edge detected (true) or not yet detected (false) on correspondind XBARA\_OUT[\*] output.

**43.2.3.12 void XBARB\_HAL\_Init ( XBARB\_Type \* *baseAddr* )**

## Parameters

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

**43.2.3.13 static void XBARB\_HAL\_SetOutSel ( XBARB\_Type \* *baseAddr*, uint32\_t *outIndex*, uint32\_t *input* ) [inline], [static]**

This function selects which of the shared inputs XBARB\_IN[\*] is muxed to selected output XBARB\_OUT[\*]

## Parameters

|                 |                                                    |
|-----------------|----------------------------------------------------|
| <i>baseAddr</i> | Register base address for XBARB module.            |
| <i>outIndex</i> | Selected XBARB_OUT[*] output.                      |
| <i>input</i>    | Input to be muxed to selected XBARB_OUT[*] output. |

**43.2.3.14 static uint32\_t XBARB\_HAL\_GetOutSel ( XBARB\_Type \* *baseAddr*, uint32\_t *outIndex* ) [inline], [static]**

This function gets input XBARB\_IN[\*] muxed to selected output XBARB\_OUT[\*]

## XBAR HAL driver

### Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>baseAddr</i> | Register base address for XBARB module. |
| <i>outIndex</i> | Selected XBARB_OUT[*] output.           |

### Returns

Input XBARB\_IN[\*] muxed to selected XBARB\_OUT[\*] output.

## 43.3 XBAR Peripheral driver

### 43.3.1 Overview

This section describes the programming interface of the XBAR Peripheral driver.

### 43.3.2 Overview

The XBAR peripheral driver configures the XBAR (Inter-Peripheral Crossbar Switch) and handles initialization and configuration of the XBAR module.

### 43.3.3 CMP Driver model building

XBAR driver has two parts:

- Signal connection - This part interconnects input and output signals.
- Active edge feature - Some of the outputs provides active edge detection. If an active edge occurs, an interrupt or a DMA request can be called. APIs handle user callbacks for the interrupts. The driver also includes API for clearing and reading status bit.

### 43.3.4 Initialization

To initialize the XBAR driver, a state structure has to be passed into the initialization function. This block of memory keeps pointers to user's callback functions and parameters to these functions. The XBAR module is initialized by calling the [XBAR\\_DRV\\_Init\(\)](#) function.

### 43.3.5 Call diagram

1. Call the "XBAR\_DRV\_Init()" function to initialize the XBAR module. A state structure of the "xbar\_state\_t" type has to be defined and passed into this function.
2. Optionally, call the "XBAR\_DRV\_ConfigOutControl()" function to set the active edge features, such interrupts or DMA requests. A configuration structure of the "xbar\_control\_config\_t" type is required.
3. Optionally, call the "XBAR\_DRV\_InstallCallback" function to install user's callback function. A pointer to the function of the "xbar\_callback\_t" type and pointer to a void parameter is required. When the XBAR is configured to enable the interrupt, the installed callback function is called after the interrupt event occurs.
4. Call the "XBAR\_DRV\_ConfigSignalConnection" function to connect the desired signals.
5. Finally, the XBAR works properly.

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

Interrupt Mode:

## XBAR Peripheral driver

```

// xbat_test.c //

#include <stdio.h>
#include <stdlib.h>
#include "board.h"
#include "fsl_debug_console.h"
#include "fsl_xbar_driver.h"

void configure_xbar_test_pins(void);

/* User's callback function*/
void xbar_callback(void * param);

int main (void)
{
 hardware_init();
 dbg_uart_init();

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

 configure_xbar_test_pins();

 xbar_state_t xbarStateStruct;

 /* Init the XBAR module*/
 XBAR_DRV_Init(&xbarStateStruct);

 uint32_t outIndex = 0;

 /* Configure an interrupt request on XBAR output 0 and install callback function */
 xbar_control_config_t xbarConfigStruct;
 xbarConfigStruct.activeEdge = kXbarEdgeFalling;
 xbarConfigStruct.intDmaReq = kXbarReqIen;

 char msg[] = "The button is pressed";

 XBAR_DRV_ConfigOutControl(outIndex, &xbarConfigStruct);
 XBAR_DRV_InstallCallback(outIndex, xbar_callback, msg);

 /* Configure interperipheral signal connections. */
 XBAR_DRV_ConfigSignalConnection(kXbaraInputXBAR_IN2, kXbaraOutputXB_OUT6
);
 XBAR_DRV_ConfigSignalConnection(kXbaraInputXBAR_IN2,
 kXbaraOutputDMAreqOrInt);

 while(1);

 XBAR_DRV_Deinit();
 printf("XBAR PD Test ");
 printf("Succeed\r\n");
}

void xbar_callback(void * param){
 char * strin = (char *)param;
 printf("%s\n", strin);
}

void configure_xbar_test_pins(void){
 PORT_HAL_SetMuxMode(PORTF_BASE, 1u, kPortMuxAlt3);
 PORT_HAL_SetMuxMode(PORTD_BASE, 0u, kPortMuxAlt3);
}

```

### Polling Mode:

```

// xbar_test_polling.c //

#include <stdio.h>
#include <stdlib.h>
#include "board.h"
#include "fsl_debug_console.h"
#include "fsl_xbar_driver.h"

void configure_xbar_test_pins(void);

int main (void)
{
 hardware_init();
 dbg_uart_init();

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

 configure_xbar_test_pins();

 xbar_state_t xbarStateStruct;

 /* Init the XBAR module*/
 XBAR_DRV_Init(&xbarStateStruct);

 uint32_t outIndex = 0;

 /* Configure an interrupt request on XBAR output 0 */
 xbar_control_config_t xbarConfigStruct;
 xbarConfigStruct.activeEdge = kXbarEdgeRising;
 xbarConfigStruct.intDmaReq = kXbarReqDis;
 XBAR_DRV_ConfigOutControl(outIndex, &xbarConfigStruct);

 /* Configure interperipheral signal connections. */
 XBAR_DRV_ConfigSignalConnection(kXbaraInputXBAR_IN2, kXbaraOutputXB_OUT6
);
 XBAR_DRV_ConfigSignalConnection(kXbaraInputXBAR_IN2,
 kXbaraOutputDMAreqOrInt);

 while(1)
 {
 if(XBAR_DRV_GetEdgeDetectionStatus(outIndex))
 {
 printf("The button is pressed\n");
 XBAR_DRV_ClearEdgeDetectionStatus(outIndex);
 }
 }

 XBAR_DRV_Deinit();
 printf("XBAR PD Test ");
 printf("Succeed\r\n");
}

void configure_xbar_test_pins(void){
 PORT_HAL_SetMuxMode(PORTF_BASE, 1u, kPortMuxAlt3);
 PORT_HAL_SetMuxMode(PORTD_BASE, 0u, kPortMuxAlt3);
}

```

## Data Structures

- struct [xbar\\_control\\_config\\_t](#)

## XBAR Peripheral driver

- Defines the configuration structure of the XBAR control register. [More...](#)*
- struct [xbar\\_state\\_t](#)  
*Internal driver state information. [More...](#)*

## Macros

- #define [XBARA\\_MODULE](#) 0U  
*Macro defines XBARA module number.*
- #define [XBARB\\_MODULE](#) 1U  
*Macro defines XBARB module number.*

## Typedefs

- typedef void(\* [xbar\\_callback\\_t](#))(void \*param)  
*Defines the type of the user-defined callback function.*

## Enumerations

- enum [xbar\\_ien\\_den\\_req\\_t](#) {  
[kXbarReqDis](#) = 0U,  
[kXbarReqIen](#) = 1U,  
[kXbarReqDen](#) = 2U }  
*Defines the XBAR DMA and interrupt configurations.*

## Functions

- [xbar\\_status\\_t](#) [XBAR\\_DRV\\_Init](#) ([xbar\\_state\\_t](#) \*xbarStatePtr)  
*Initializes the XBAR modules.*
- void [XBAR\\_DRV\\_Deinit](#) (void)  
*De-initializes the XBAR module.*
- [xbar\\_status\\_t](#) [XBAR\\_DRV\\_ConfigSignalConnection](#) ([xbar\\_input\\_signal\\_t](#) input, [xbar\\_output\\_signal\\_t](#) output)  
*Configures connection between the selected XBAR\_IN[\*] input and the XBAR\_OUT[\*] output signal.*
- [xbar\\_status\\_t](#) [XBAR\\_DRV\\_ConfigOutControl](#) (uint32\_t outIndex, const [xbar\\_control\\_config\\_t](#) \*controlConfigPtr)  
*Configures the XBAR control register.*
- bool [XBAR\\_DRV\\_GetEdgeDetectionStatus](#) (uint32\_t outIndex)  
*Gets the active edge detection status.*
- [xbar\\_status\\_t](#) [XBAR\\_DRV\\_ClearEdgeDetectionStatus](#) (uint32\_t outIndex)  
*Clears the status flag of the edge detection status flag for a desired XBAR\_OUT.*
- [xbar\\_status\\_t](#) [XBAR\\_DRV\\_InstallCallback](#) (uint32\_t outIndex, [xbar\\_callback\\_t](#) userCallback, void \*callbackParam)  
*Installs the callback function for the XBAR module.*
- void [XBAR\\_DRV\\_IRQHandler](#) (void)  
*Driver-defined ISR in XBAR module.*



## Variables

- XBARA\_Type \*const [g\\_xbaraBase](#) []  
*Table of base addresses for XBAR instances.*
- const IRQn\_Type [g\\_xbarIrqId](#) []  
*Table to save XBAR IRQ enumeration numbers defined in the CMSIS header file.*
- [xbar\\_active\\_edge\\_t](#) [xbar\\_control\\_config\\_t::activeEdge](#)  
*Active edge to be detected.*
- [xbar\\_ien\\_den\\_req\\_t](#) [xbar\\_control\\_config\\_t::intDmaReq](#)  
*Selects DMA/Interrupt request.*
- [xbar\\_callback\\_t](#) [xbar\\_state\\_t::userCallbackFunc](#) [FSL\_FEATURE\_XBARA\_INTERRUPT\_COUNT]  
*Keep the user-defined callback function.*
- void \* [xbar\\_state\\_t::xbarCallbackParam](#) [FSL\_FEATURE\_XBARA\_INTERRUPT\_COUNT]  
*XBAR callback parameter pointer.*

## 43.3.6 Data Structure Documentation

### 43.3.6.1 struct [xbar\\_control\\_config\\_t](#)

This structure keeps the configuration of XBAR control register for one output. Control registers are available only for a few outputs. Not every XBAR module has control registers.

#### Data Fields

- [xbar\\_active\\_edge\\_t](#) [activeEdge](#)  
*Active edge to be detected.*
- [xbar\\_ien\\_den\\_req\\_t](#) [intDmaReq](#)  
*Selects DMA/Interrupt request.*

### 43.3.6.2 struct [xbar\\_state\\_t](#)

The contents of this structure are internal to the driver and should not be modified by users.

#### Data Fields

- [xbar\\_callback\\_t](#) [userCallbackFunc](#) [FSL\_FEATURE\_XBARA\_INTERRUPT\_COUNT]  
*Keep the user-defined callback function.*
- void \* [xbarCallbackParam](#) [FSL\_FEATURE\_XBARA\_INTERRUPT\_COUNT]  
*XBAR callback parameter pointer.*

## XBAR Peripheral driver

### 43.3.7 Typedef Documentation

#### 43.3.7.1 typedef void(\* xbar\_callback\_t)(void \*param)

A prototype for the callback function registered into the XBAR driver.

### 43.3.8 Enumeration Type Documentation

#### 43.3.8.1 enum xbar\_ien\_den\_req\_t

Defines the XBAR DMA and interrupt configurations.

Enumerator

- kXbarReqDis* Interrupt and DMA are disabled.
- kXbarReqIen* Interrupt enabled, DMA disabled.
- kXbarReqDen* DMA enabled, interrupt disabled.

### 43.3.9 Function Documentation

#### 43.3.9.1 xbar\_status\_t XBAR\_DRV\_Init ( xbar\_state\_t \* xbarStatePtr )

This function initializes the XBAR module to a reset state.

Parameters

|                     |                    |
|---------------------|--------------------|
| <i>xbarStatePtr</i> | XBAR input signal. |
|---------------------|--------------------|

Returns

An error code or kStatus\_XBAR\_Success.

#### 43.3.9.2 void XBAR\_DRV\_Deinit ( void )

This function clears all configurations and shuts down the XBAR by disabling interrupt and the clock signal to the modules.

#### 43.3.9.3 xbar\_status\_t XBAR\_DRV\_ConfigSignalConnection ( xbar\_input\_signal\_t *input*, xbar\_output\_signal\_t *output* )

This function configures which XBAR input is connected to the selected XBAR output. If more than one XBAR module is available, only the inputs and outputs from the same module can be connected.

Example:

```
xbar_status_t status;

// Configure connection between XBARA_OUT16(CMP0) output and XBARA_IN1(VDD) input.
status = XBARA_HAL_ConfigSignalConnection(kXbaraInputVDD, kXbaraOutputCMP0);
switch (status)
{
 //...
}
```

Parameters

|               |                     |
|---------------|---------------------|
| <i>input</i>  | XBAR input signal.  |
| <i>output</i> | XBAR output signal. |

Returns

An error code or kStatus\_XBAR\_Success.

### 43.3.9.4 `xbar_status_t XBAR_DRV_ConfigOutControl ( uint32_t outIndex, const xbar_control_config_t * controlConfigPtr )`

This function configures an XBAR control register. The active edge detection and the DMA/IRQ function on the corresponding XBAR output can be set.

Example:

```
xbar_status_t status;

// Set the DMA function on XBAR_OUT2 to rising and falling active edges.
xbar_control_config_t controlOut2;

controlOut2.activeEdge = kXbarEdgeRisingAndFalling;
controlOut2.intDmaReq = kXbarReqDen;

status = XBAR_DRV_ConfigOutControl(2, controlOut2);
switch (status)
{
 //...
}
```

Parameters

## XBAR Peripheral driver

|                         |                                                                    |
|-------------------------|--------------------------------------------------------------------|
| <i>outIndex</i>         | XBAR output number.                                                |
| <i>controlConfigPtr</i> | Pointer to structure that keeps configuration of control register. |

Returns

An error code or `kStatus_XBAR_Success`.

### 43.3.9.5 `bool XBAR_DRV_GetEdgeDetectionStatus ( uint32_t outIndex )`

This function gets the active edge detect status of the desired XBAR\_OUT. If the active edge occurs, the return value is asserted. When the interrupt or the DMA functionality is enabled for the XBAR\_OUTx, this field is 1 when the interrupt or DMA request is asserted and 0 when the interrupt or DMA request has been cleared.

Parameters

|                 |                     |
|-----------------|---------------------|
| <i>outIndex</i> | XBAR output number. |
|-----------------|---------------------|

Returns

Assertion of edge detection status.

### 43.3.9.6 `xbar_status_t XBAR_DRV_ClearEdgeDetectionStatus ( uint32_t outIndex )`

This function clears the status flag of edge detection status flag of the XBAR\_OUTx.

Parameters

|                 |                     |
|-----------------|---------------------|
| <i>outIndex</i> | XBAR output number. |
|-----------------|---------------------|

Returns

An error code or `kStatus_XBAR_Success`.

### 43.3.9.7 `xbar_status_t XBAR_DRV_InstallCallback ( uint32_t outIndex, xbar_callback_t userCallback, void * callbackParam )`

This function installs the user-defined callback. When the XBAR interrupt request is configured, the callback is executed inside the ISR.

## Parameters

|                      |                                      |
|----------------------|--------------------------------------|
| <i>userCallback</i>  | User-defined callback function.      |
| <i>callbackParam</i> | The XBAR callback parameter pointer. |

## Returns

An error code or kStatus\_XBAR\_Success.

**43.3.9.8 void XBAR\_DRV\_IRQHandler ( void )**

This function is the driver-defined ISR in XBAR module. It includes the process for interrupt mode defined by the driver. Currently, it is called inside the system-defined ISR.

**43.3.10 Variable Documentation**

**43.3.10.1 XBARA\_Type\* const g\_xbaraBase[]**

**43.3.10.2 const IRQn\_Type g\_xbarIrqId[]**

**43.3.10.3 xbar\_active\_edge\_t xbar\_control\_config\_t::activeEdge**

**43.3.10.4 xbar\_ien\_den\_req\_t xbar\_control\_config\_t::intDmaReq**

**43.3.10.5 xbar\_callback\_t xbar\_state\_t::userCallbackFunc[FSL\_FEATURE\_XBARA\_INTERRUPT\_COUNT]**

**43.3.10.6 void\* xbar\_state\_t::xbarCallbackParam[FSL\_FEATURE\_XBARA\_INTERRUPT\_COUNT]**



## Chapter 44 Low-Leakage Wakeup Unit (LLWU)

### 44.1 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 configure external wakeup pin and internal wakeup modules as wake-up source from low-leakage power modes.

### 44.2 External Wakeup Pin APIs

Different platforms may have different wakeup pin assignments, please check reference manual for details. The external wakeup pin could be configured by the function `LLWU_HAL_SetExternalInputPinMode()`.

After wakeup, there are flags indicate which external pin is the wakeup source, please use the function `LLWU_HAL_GetExternalPinWakeupFlag()` to check the external pin wakeup flags. The function `LLWU_HAL_ClearExternalPinWakeupFlag()` could clear the wakeup flags.

For external pin, there is pin filter, the function `LLWU_HAL_SetPinFilterMode()` configure the pin filter for specific pin and configure the filter mode.

### 44.3 Internal Wakeup Module APIs

Different platforms may have different wakeup module assignments, please check reference manual for details. The internal wakeup module could be enable/disable by the function `LLWU_HAL_SetInternalModuleCmd()`.

After wakeup, there are flags indicate which module is the wakeup source, please use the function `LLWU_HAL_GetInternalModuleWakeupFlag()` to check the internal module wakeup flags. There is no API to clear internal wakeup module flags, please clear the wakeup source status directly.

### 44.4 Reset Pin Configure APIs

The RESET pin can be configured as low-leakage mode exit source, the function `LLWU_HAL_SetResetPinMode()` set this feature, also the RESET pin filter could be configured by this function.

#### 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\\_pin\\_mode\\_t](#)  
*Reset pin control structure. [More...](#)*

## Enumeration Type Documentation

### Enumerations

- enum `llwu_external_pin_modes_t` {  
`kLlwuExternalPinDisabled`,  
`kLlwuExternalPinRisingEdge`,  
`kLlwuExternalPinFallingEdge`,  
`kLlwuExternalPinChangeDetect` }  
*External input pin control modes.*
- enum `llwu_filter_modes_t` {  
`kLlwuFilterDisabled`,  
`kLlwuFilterPosEdgeDetect`,  
`kLlwuFilterNegEdgeDetect`,  
`kLlwuFilterAnyEdgeDetect` }  
*Digital filter control modes.*

## 44.5 Data Structure Documentation

### 44.5.1 struct `llwu_external_pin_filter_mode_t`

#### Data Fields

- `llwu_filter_modes_t filterMode`  
*Filter mode.*
- `llwu_wakeup_pin_t pinNumber`  
*Pin number.*

### 44.5.2 struct `llwu_reset_pin_mode_t`

#### Data Fields

- bool `enable`  
*RESET pin is enabled as low-leakage mode exit source.*
- bool `filter`  
*Digital filter on RESET pin.*

#### 44.5.2.0.0.72 Field Documentation

44.5.2.0.0.72.1 bool `llwu_reset_pin_mode_t::enable`

44.5.2.0.0.72.2 bool `llwu_reset_pin_mode_t::filter`

## 44.6 Enumeration Type Documentation

### 44.6.1 enum `llwu_external_pin_modes_t`

Enumerator

***kLlwuExternalPinDisabled*** Pin disabled as wakeup input.



*kLlwuExternalPinRisingEdge* Pin enabled with rising edge detection.  
*kLlwuExternalPinFallingEdge* Pin enabled with falling edge detection.  
*kLlwuExternalPinChangeDetect* Pin enabled with any change detection.

#### 44.6.2 enum llwu\_filter\_modes\_t

Enumerator

*kLlwuFilterDisabled* Filter disabled.  
*kLlwuFilterPosEdgeDetect* Filter positive edge detection.  
*kLlwuFilterNegEdgeDetect* Filter negative edge detection.  
*kLlwuFilterAnyEdgeDetect* Filter any edge detection.



---

## Enumeration Type Documentation

---

## Chapter 45

# Multipurpose Clock Generator (MCG)

### 45.1 Overview

The Kinetis SDK provides a HAL driver for the Multipurpose Clock Generator (MCG) block of Kinetis devices.

### Modules

- [MCG HAL driver](#)

## MCG HAL driver

### 45.2 MCG HAL driver

#### 45.2.1 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;

#### 45.2.2 OSC related APIs

MCG uses the OSC as an external reference clock source. To use OSC, please set RANGE, HGO and EREFS correctly according to the board configuration. The functions [CLOCK\\_HAL\\_SetOsc0Mode\(\)](#)/[CLOCK\\_HAL\\_SetOsc1Mode\(\)](#) are used for this purpose.

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.

MCG provides monitor for each OSC, when OSC clock lost, the monitor could trigger interrupt or system reset. To enable OSC monitor, use functions [CLOCK\\_HAL\\_EnableXXXMonitor\(\)](#), here XXX is the OSC name, such as `Osc0`, `Osc1`, `RtcOsc`.

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.

#### 45.2.3 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. The FLL reference clock divider FRDIV should be configured to output a reference clock in the range of 31.25kHz to 39.0625kHz, please check reference manual for more details. For easy use, the function [CLOCK\\_HAL\\_GetAvailableFrdiv\(\)](#) could help to get the proper FRDIV value.

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.

#### 45.2.4 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.

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.

PLL could be enabled when MCG is in FLL mode, please use the functions [CLOCK\\_HAL\\_EnablePlloInFllMode\(\)](#) and [CLOCK\\_HAL\\_EnablePlliInFllMode\(\)](#) to enable PLL in FLL mode.

For some platforms, PLL loss of lock could trigger system reset, please use the function [CLOCK\\_HAL\\_SetPllostLockResetCmd\(\)](#) to configure this feature.

#### 45.2.5 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.

#### 45.2.6 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 function [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.

The MCG ATM has a flag indicates the trim failed or not. Please use the functions [CLOCK\\_HAL\\_](#)

## MCG HAL driver

[IsAutoTrimMachineFailed\(\)](#) and [CLOCK\\_HAL\\_ClearAutoTrimMachineFailed\(\)](#) to check and clear the flag.

## Files

- file [fsl\\_mcg\\_hal.h](#)

## Enumerations

- enum [\\_mcg\\_constant](#)  
*MCG constant definitions.*
- enum [mcg\\_fl\\_src\\_t](#) {  
[kMcgFlSrcExternal](#),  
[kMcgFlSrcInternal](#) }  
*MCG internal reference clock source select.*
- enum [osc\\_range\\_t](#) {  
[kOscRangeLow](#),  
[kOscRangeHigh](#),  
[kOscRangeVeryHigh](#),  
[kOscRangeVeryHigh1](#),  
[kOscRangeLow](#),  
[kOscRangeHigh](#),  
[kOscRangeVeryHigh](#),  
[kOscRangeVeryHigh1](#) }  
*MCG OSC frequency range select.*
- enum [osc\\_gain\\_t](#) {  
[kOscGainLow](#),  
[kOscGainHigh](#),  
[kOscGainLow](#),  
[kOscGainHigh](#) }  
*MCG high gain oscillator select.*
- enum [osc\\_src\\_t](#) {  
[kOscSrcExt](#),  
[kOscSrcOsc](#),  
[kOscSrcExt](#),  
[kOscSrcOsc](#) }  
*MCG external reference clock select.*
- enum [mcg\\_irc\\_mode\\_t](#) {  
[kMcgIrcSlow](#),  
[kMcgIrcFast](#) }  
*MCG internal reference clock select.*
- enum [mcg\\_dmx32\\_select\\_t](#) {  
[kMcgDmx32Default](#),  
[kMcgDmx32Fine](#) }  
*MCG DCO Maximum Frequency with 32.768 kHz Reference.*

- enum `mcg_dco_range_select_t` {  
`kMcgDcoRangeSelLow`,  
`kMcgDcoRangeSelMid`,  
`kMcgDcoRangeSelMidHigh`,  
`kMcgDcoRangeSelHigh` }  
*MCG DCO range select.*
- enum `mcg_pll_ref_mode_t` {  
`kMcgPllRefOsc0`,  
`kMcgPllRefOsc1` }  
*MCG PLL external reference clock select.*
- enum `mcg_clkout_src_t` {  
`kMcgClkOutSrcOut`,  
`kMcgClkOutSrcInternal`,  
`kMcgClkOutSrcExternal` }  
*MCGOUT clock source.*
- enum `mcg_clkout_stat_t` {  
`kMcgClkOutStatFll`,  
`kMcgClkOutStatInternal`,  
`kMcgClkOutStatExternal`,  
`kMcgClkOutStatPll` }  
*MCG clock mode status.*
- enum `mcg_atm_select_t` {  
`kMcgAtmSel32k`,  
`kMcgAtmSel4m` }  
*MCG Automatic Trim Machine Select.*
- enum `mcg_oscsel_select_t` {  
`kMcgOscselOsc`,  
`kMcgOscselRtc` }  
*MCG OSC Clock Select.*
- enum `mcg_osc_monitor_mode_t` {  
`kMcgOscMonitorInt`,  
`kMcgOscMonitorReset` }  
*MCG OSC monitor mode.*
- enum `mcg_pll_clk_select_t` { `kMcgPllClkSelPll0` }  
*MCG PLLCS select.*
- enum `mcg_atm_error_t` {  
`kMcgAtmErrorNone`,  
`kMcgAtmErrorBusClockRange`,  
`kMcgAtmErrorDesireFreqRange`,  
`kMcgAtmErrorIrcUsed`,  
`kMcgAtmErrorTrimValueInvalid`,  
`kMcgAtmErrorHardwareFail` }  
*MCG auto trim machine error code.*
- enum `mcg_status_t` {  
`kStatus_MCG_Success` = 0U,  
`kStatus_MCG_Fail` = 1U }  
*MCG status.*

## MCG HAL driver

- enum `mcg_modes_t` {
  - `kMcgModeFEI` = 0x01 << 0U,
  - `kMcgModeFBI` = 0x01 << 1U,
  - `kMcgModeBLPI` = 0x01 << 2U,
  - `kMcgModeFEE` = 0x01 << 3U,
  - `kMcgModeFBE` = 0x01 << 4U,
  - `kMcgModeBLPE` = 0x01 << 5U,
  - `kMcgModePBE` = 0x01 << 6U,
  - `kMcgModePEE` = 0x01 << 7U,
  - `kMcgModeSTOP` = 0x01 << 8U,
  - `kMcgModeError` = 0x01 << 9U }

*MCG mode definitions.*
- enum `mcg_mode_error_t` {
  - `kMcgModeErrNone` = 0x00U,
  - `kMcgModeErrModeUnreachable` = 0x01U,
  - `kMcgModeErrOscFreqRange` = 0x21U,
  - `kMcgModeErrIrcSlowRange` = 0x31U,
  - `kMcgModeErrIrcFastRange` = 0x32U,
  - `kMcgModeErrFllRefRange` = 0x33U,
  - `kMcgModeErrFllFrdivRange` = 0x34U,
  - `kMcgModeErrFllDrsRange` = 0x35U,
  - `kMcgModeErrFllDmx32Range` = 0x36U,
  - `kMcgModeErrPllPrdivRange` = 0x41U,
  - `kMcgModeErrPllVdivRange` = 0x42U,
  - `kMcgModeErrPllRefClkRange` = 0x43U,
  - `kMcgModeErrPllLockBit` = 0x44U,
  - `kMcgModeErrPllOutClkRange` = 0x45U }

*MCG mode transition API error code definitions.*

## Functions

- `mcg_modes_t` `CLOCK_HAL_GetMcgMode` (MCG\_Type \*base)
  - Gets the current MCG mode.*
- `mcg_mode_error_t` `CLOCK_HAL_SetFeiMode` (MCG\_Type \*base, `mcg_dco_range_select_t` drs, void(\*fllStableDelay)(void), uint32\_t \*outClkFreq)
  - Set MCG to FEI mode.*
- `mcg_mode_error_t` `CLOCK_HAL_SetFeeMode` (MCG\_Type \*base, `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` (MCG\_Type \*base, `mcg_dco_range_select_t` drs, `mcg_irc_mode_t` ircSelect, uint8\_t frdivVal, void(\*fllStableDelay)(void), uint32\_t \*outClkFreq)
  - Set MCG to FBI mode.*
- `mcg_mode_error_t` `CLOCK_HAL_SetFbeMode` (MCG\_Type \*base, `mcg_oscsel_select_t` oscselVal, uint8\_t frdivVal, `mcg_dmx32_select_t` dmx32, `mcg_dco_range_select_t` drs, void(\*fllStable-



- Delay)(void), uint32\_t \*outClkFreq)
  - Set MCG to FBE mode.*
- [mcg\\_mode\\_error\\_t CLOCK\\_HAL\\_SetBlpiMode](#) (MCG\_Type \*base, uint8\_t fcrdivVal, [mcg\\_irc\\_mode\\_t](#) ircSelect, uint32\_t \*outClkFreq)
  - Set MCG to BLPI mode.*
- [mcg\\_mode\\_error\\_t CLOCK\\_HAL\\_SetBlpeMode](#) (MCG\_Type \*base, [mcg\\_oscsel\\_select\\_t](#) oscselVal, uint32\_t \*outClkFreq)
  - Set MCG to BLPE mode.*
- [mcg\\_mode\\_error\\_t CLOCK\\_HAL\\_SetPbeMode](#) (MCG\_Type \*base, [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](#) (MCG\_Type \*base, uint32\_t \*outClkFreq)
  - Set MCG to PBE mode.*

## MCG out clock access API

- uint32\_t [CLOCK\\_HAL\\_TestOscFreq](#) (MCG\_Type \*base, [mcg\\_oscsel\\_select\\_t](#) oscselVal)
  - Tests the external clock frequency.*
- uint32\_t [CLOCK\\_HAL\\_TestFllExternalRefFreq](#) (MCG\_Type \*base, uint32\_t extFreq, uint8\_t frdivVal, [osc\\_range\\_t](#) range0, [mcg\\_oscsel\\_select\\_t](#) oscsel)
  - Tests the FLL external reference frequency based on the input parameters.*
- uint32\_t [CLOCK\\_HAL\\_GetFllRefClk](#) (MCG\_Type \*base)
  - Gets the current MCG FLL clock.*
- uint32\_t [CLOCK\\_HAL\\_TestFllFreq](#) (MCG\_Type \*base, uint32\_t fllRef, [mcg\\_dmx32\\_select\\_t](#) dmx32, [mcg\\_dco\\_range\\_select\\_t](#) drs)
  - Calculates the FLL frequency based on the input parameters.*
- uint32\_t [CLOCK\\_HAL\\_GetFllClk](#) (MCG\_Type \*base)
  - Gets the current MCG FLL clock.*
- uint32\_t [CLOCK\\_HAL\\_GetInternalRefClk](#) (MCG\_Type \*base)
  - Gets the current MCG internal reference clock(MCGIRCLK).*
- uint32\_t [CLOCK\\_HAL\\_GetFixedFreqClk](#) (MCG\_Type \*base)
  - Gets the current MCG fixed frequency clock(MCGFFCLK).*
- uint32\_t [CLOCK\\_HAL\\_GetOutClk](#) (MCG\_Type \*base)
  - Gets the current MCG out clock.*

## MCG control register access API

- static void [CLOCK\\_HAL\\_SetClkOutSrc](#) (MCG\_Type \*base, [mcg\\_clkout\\_src\\_t](#) select)
  - Sets the Clock Source Select.*
- static [mcg\\_clkout\\_stat\\_t](#) [CLOCK\\_HAL\\_GetClkOutStat](#) (MCG\_Type \*base)
  - Gets the Clock Mode Status.*
- static void [CLOCK\\_HAL\\_SetLowPowerModeCmd](#) (MCG\_Type \*base, bool enable)
  - Sets the Low Power Select.*

## MCG HAL driver

### MCG FLL API

- static `mcg_fl_src_t CLOCK_HAL_GetFlSrc` (MCG\_Type \*base)  
*Gets the FLL source status.*
- static void `CLOCK_HAL_SetFlFilterPreserveCmd` (MCG\_Type \*base, bool enable)  
*Sets the FLL Filter Preserve Enable Setting.*
- `mcg_status_t CLOCK_HAL_GetAvailableFrdiv` (`osc_range_t` range0, `mcg_oscsel_select_t` oscsel, `uint32_t` inputFreq, `uint8_t` \*frdiv)  
*Calculates the proper FRDIV setting.*

### MCG internal reference clock APIs

- static void `CLOCK_HAL_SetInternalRefClkEnableCmd` (MCG\_Type \*base, bool enable)  
*Sets the internal reference clock enable or not.*
- static void `CLOCK_HAL_SetInternalRefClkEnableInStopCmd` (MCG\_Type \*base, bool enable)  
*Sets the internal reference clock enable or nor in stop mode.*
- static void `CLOCK_HAL_SetInternalRefClkMode` (MCG\_Type \*base, `mcg_irc_mode_t` mode)  
*Sets the Internal Reference Clock Select.*
- static `mcg_irc_mode_t` `CLOCK_HAL_GetInternalRefClkMode` (MCG\_Type \*base)  
*Gets the Internal Reference Clock Status.*
- void `CLOCK_HAL_UpdateFastClkInternalRefDiv` (MCG\_Type \*base, `uint8_t` fcrdiv)  
*Updates the Fast Clock Internal Reference Divider Setting.*

### MCG OSC APIs

- void `CLOCK_HAL_SetOsc0Mode` (MCG\_Type \*base, `osc_range_t` range, `osc_gain_t` hgo, `osc_src_t` erefs)  
*Sets the OSC0 work mode.*
- static bool `CLOCK_HAL_IsOsc0Stable` (MCG\_Type \*base)  
*Gets the OSC Initialization Status.*
- void `CLOCK_HAL_EnableOsc0Monitor` (MCG\_Type \*base, `mcg_osc_monitor_mode_t` mode)  
*Enables the OSC0 external clock monitor.*
- static void `CLOCK_HAL_DisableOsc0Monitor` (MCG\_Type \*base)  
*Disables the OSC0 external clock monitor.*
- static bool `CLOCK_HAL_IsOsc0MonitorEnabled` (MCG\_Type \*base)  
*Checks the OSC0 external clock monitor is enabled or not.*
- static bool `CLOCK_HAL_IsOsc0LostLock` (MCG\_Type \*base)  
*Gets the OSC0 Loss of Clock Status.*
- static void `CLOCK_HAL_ClearOsc0LostLock` (MCG\_Type \*base)  
*Clears the OSC0 Loss of Clock Status.*

### MCG Auto Trim Machine (ATM)

- `mcg_atm_error_t` `CLOCK_HAL_TrimInternalRefClk` (MCG\_Type \*base, `uint32_t` extFreq, `uint32_t` desireFreq, `uint32_t` \*actualFreq, `mcg_atm_select_t` atms)  
*Auto trims the internal reference clock.*

- static bool `CLOCK_HAL_IsAutoTrimMachineFailed` (MCG\_Type \*base)  
*Gets the Automatic Trim machine Fail Flag.*
- static void `CLOCK_HAL_ClearAutoTrimMachineFailed` (MCG\_Type \*base)  
*Clears the Automatic Trim machine Fail Flag.*

## 45.2.7 Enumeration Type Documentation

### 45.2.7.1 enum mcg\_fl\_src\_t

Enumerator

- kMcgFlSrcExternal*** External reference clock is selected.
- kMcgFlSrcInternal*** The slow internal reference clock is selected.

### 45.2.7.2 enum osc\_range\_t

Enumerator

- kOscRangeLow*** Low frequency range selected for the crystal OSC.
- kOscRangeHigh*** High frequency range selected for the crystal OSC.
- kOscRangeVeryHigh*** Very High frequency range selected for the crystal OSC.
- kOscRangeVeryHigh1*** Very High frequency range selected for the crystal OSC.
- kOscRangeLow*** Low frequency range selected for the crystal OSC.
- kOscRangeHigh*** High frequency range selected for the crystal OSC.
- kOscRangeVeryHigh*** Very High frequency range selected for the crystal OSC.
- kOscRangeVeryHigh1*** Very High frequency range selected for the crystal OSC.

### 45.2.7.3 enum osc\_gain\_t

Enumerator

- kOscGainLow*** Configure crystal oscillator for low-power operation.
- kOscGainHigh*** Configure crystal oscillator for high-gain operation.
- kOscGainLow*** Configure crystal oscillator for low-power operation.
- kOscGainHigh*** Configure crystal oscillator for high-gain operation.

### 45.2.7.4 enum osc\_src\_t

Enumerator

- kOscSrcExt*** External reference clock requested.
- kOscSrcOsc*** Oscillator requested.
- kOscSrcExt*** Selects external input clock.
- kOscSrcOsc*** Selects Oscillator.

## MCG HAL driver

### 45.2.7.5 enum mcg\_irc\_mode\_t

Enumerator

*kMcgIrcSlow* Slow internal reference clock selected.

*kMcgIrcFast* Fast internal reference clock selected.

### 45.2.7.6 enum mcg\_dmx32\_select\_t

Enumerator

*kMcgDmx32Default* DCO has a default range of 25%.

*kMcgDmx32Fine* DCO is fine-tuned for maximum frequency with 32.768 kHz reference.

### 45.2.7.7 enum mcg\_dco\_range\_select\_t

Enumerator

*kMcgDcoRangeSelLow* Low frequency range.

*kMcgDcoRangeSelMid* Mid frequency range.

*kMcgDcoRangeSelMidHigh* Mid-High frequency range.

*kMcgDcoRangeSelHigh* High frequency range.

### 45.2.7.8 enum mcg\_pll\_ref\_mode\_t

Enumerator

*kMcgPllRefOsc0* Selects OSC0 clock source as its external reference clock.

*kMcgPllRefOsc1* Selects OSC1 clock source as its external reference clock.

### 45.2.7.9 enum mcg\_clkout\_src\_t

Enumerator

*kMcgClkOutSrcOut* Output of the FLL is selected (reset default)

*kMcgClkOutSrcInternal* Internal reference clock is selected.

*kMcgClkOutSrcExternal* External reference clock is selected.

#### 45.2.7.10 enum mcg\_clkout\_stat\_t

Enumerator

- kMcgClkOutStatFll* Output of the FLL is selected (reset default)
- kMcgClkOutStatInternal* Internal reference clock is selected.
- kMcgClkOutStatExternal* External reference clock is selected.
- kMcgClkOutStatPll* Output of the PLL is selected.

#### 45.2.7.11 enum mcg\_atm\_select\_t

Enumerator

- kMcgAtmSel32k* 32 kHz Internal Reference Clock selected
- kMcgAtmSel4m* 4 MHz Internal Reference Clock selected

#### 45.2.7.12 enum mcg\_oscsel\_select\_t

Enumerator

- kMcgOscselOsc* Selects System Oscillator (OSCCLK)
- kMcgOscselRtc* Selects 32 kHz RTC Oscillator.

#### 45.2.7.13 enum mcg\_osc\_monitor\_mode\_t

Enumerator

- kMcgOscMonitorInt* Generate interrupt when clock lost.
- kMcgOscMonitorReset* Generate reset when clock lost.

#### 45.2.7.14 enum mcg\_pll\_clk\_select\_t

Enumerator

- kMcgPllClkSelPll0* PLL0 output clock is selected.

#### 45.2.7.15 enum mcg\_atm\_error\_t

Enumerator

- kMcgAtmErrorNone* No error.
- kMcgAtmErrorBusClockRange* Bus clock frequency is not in 8MHz to 16 MHz.

## MCG HAL driver

***kMcgAtmErrorDesireFreqRange*** Desired frequency is out of range.  
***kMcgAtmErrorIrcUsed*** IRC clock is used to generate system clock.  
***kMcgAtmErrorTrimValueInvalid*** The auto trim compare value ACT is invalid.  
***kMcgAtmErrorHardwareFail*** ATC[ATMF] fail flag asserts.

### 45.2.7.16 enum mcg\_status\_t

Enumerator

***kStatus\_MCG\_Success*** Success.  
***kStatus\_MCG\_Fail*** Execution failed.

### 45.2.7.17 enum mcg\_modes\_t

Enumerator

***kMcgModeFEI*** FEI - FLL Engaged Internal.  
***kMcgModeFBI*** FBI - FLL Bypassed Internal.  
***kMcgModeBLPI*** BLPI - Bypassed Low Power Internal.  
***kMcgModeFEE*** FEE - FLL Engaged External.  
***kMcgModeFBE*** FBE - FLL Bypassed External.  
***kMcgModeBLPE*** BLPE - Bypassed Low Power External.  
***kMcgModePBE*** PBE - PLL Bypassed External.  
***kMcgModePEE*** PEE - PLL Engaged External.  
***kMcgModeSTOP*** STOP - Stop.  
***kMcgModeError*** Unknown mode.

### 45.2.7.18 enum mcg\_mode\_error\_t

Enumerator

***kMcgModeErrNone*** No error.  
***kMcgModeErrModeUnreachable*** Target mode is unreachable.  
***kMcgModeErrOscFreqRange*** OSC frequency is invalid.  
***kMcgModeErrIrcSlowRange*** slow IRC is outside allowed range  
***kMcgModeErrIrcFastRange*** fast IRC is outside allowed range  
***kMcgModeErrFllRefRange*** FLL reference frequency is outside allowed range.  
***kMcgModeErrFllFrdivRange*** FRDIV outside allowed range.  
***kMcgModeErrFllDrsRange*** DRS is out of range.  
***kMcgModeErrFllDmx32Range*** DMX32 setting not allowed.  
***kMcgModeErrPllPrdivRange*** PRDIV outside allowed range.  
***kMcgModeErrPllVdivRange*** VDIV outside allowed range.  
***kMcgModeErrPllRefClkRange*** PLL reference clock frequency, out of range.

*kMcgModeErrPllLockBit* LOCK or LOCK2 bit did not set.  
*kMcgModeErrPllOutClkRange* PLL output frequency is outside allowed range.

## 45.2.8 Function Documentation

### 45.2.8.1 `uint32_t CLOCK_HAL_TestOscFreq ( MCG_Type * base, mcg_oscsel_select_t oscselVal )`

This function tests the external clock frequency, including OSC, RTC and IRC48M. If the OSC is not initialized, this function returns 0.

Parameters

|                  |                                        |
|------------------|----------------------------------------|
| <i>base</i>      | Base address for current MCG instance. |
| <i>oscselVal</i> | External OSC selection.                |

Returns

MCG external reference clock frequency.

### 45.2.8.2 `uint32_t CLOCK_HAL_TestFllExternalRefFreq ( MCG_Type * base, uint32_t extFreq, uint8_t frdivVal, osc_range_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.

Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>base</i>     | Base address for current MCG instance.        |
| <i>extFreq</i>  | External OSC frequency.                       |
| <i>frdivVal</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.

### 45.2.8.3 `uint32_t CLOCK_HAL_GetFllRefClk ( MCG_Type * base )`

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.

## MCG HAL driver

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

### Returns

Frequency value in Hertz of FLL reference clock.

#### 45.2.8.4 `uint32_t CLOCK_HAL_TestFllFreq ( MCG_Type * base, 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>base</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

Frequency value in Hertz of the mcgflclk.

#### 45.2.8.5 `uint32_t CLOCK_HAL_GetFllClk ( MCG_Type * base )`

This function returns the mcgflclk value in frequency(Hertz) based on the current MCG configurations and settings. FLL should be properly configured get the valid value.

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

### Returns

Frequency value in Hertz of the mcgplclk.

#### 45.2.8.6 `uint32_t CLOCK_HAL_GetInternalRefClk ( MCG_Type * base )`

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>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

## Returns

Frequency value in Hertz of the MCGIRCLK.

**45.2.8.7 uint32\_t CLOCK\_HAL\_GetFixedFreqClk ( MCG\_Type \* *base* )**

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.

## Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

## Returns

Frequency value in Hertz of MCGFFCLK.

**45.2.8.8 uint32\_t CLOCK\_HAL\_GetOutClk ( MCG\_Type \* *base* )**

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>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

## Returns

Frequency value in Hertz of mcgoutclk.

**45.2.8.9 static void CLOCK\_HAL\_SetClkOutSrc ( MCG\_Type \* *base*, mcg\_clkout\_src\_t *select* ) [inline], [static]**

This function selects the clock source for the MCGOUTCLK.

## MCG HAL driver

### Parameters

|               |                                                                                                                                                                                                                                                                                   |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |

#### 45.2.8.10 `static mcg_clkout_stat_t CLOCK_HAL_GetClkOutStat ( MCG_Type * base ) [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>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

### Returns

#### 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.

#### 45.2.8.11 `static void CLOCK_HAL_SetLowPowerModeCmd ( MCG_Type * base, bool enable ) [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 effect.

### Parameters

---

|               |                                                                                                                                                                                                          |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | Base address for current MCG instance.                                                                                                                                                                   |
| <i>enable</i> | Enable low power or not: <ul style="list-style-type: none"> <li>• true: FLL (or PLL) is not disabled in bypass modes</li> <li>• false: FLL (or PLL) is disabled in bypass modes (lower power)</li> </ul> |

#### 45.2.8.12 **static mcg\_fl\_src\_t CLOCK\_HAL\_GetFlISrc ( MCG\_Type \* *base* ) [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>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

Returns

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.

#### 45.2.8.13 **static void CLOCK\_HAL\_SetFlIFilterPreserveCmd ( MCG\_Type \* *base*, 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>base</i>   | Base address for current MCG instance.                                                                                                                                                                                                                                         |
| <i>enable</i> | 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> |

## MCG HAL driver

**45.2.8.14** `mcg_status_t` **CLOCK\_HAL\_GetAvailableFrdiv** ( `osc_range_t` *range0*,  
`mcg_oscsel_select_t` *oscsel*, `uint32_t` *inputFreq*, `uint8_t` \* *frdiv* )

This function calculates the proper FRDIV setting according to the FLL reference clock. FLL reference clock frequency after FRDIV must be in the range of 31.25 kHz to 39.0625 kHz.

## 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. |

**45.2.8.15 static void CLOCK\_HAL\_SetInternalRefClkEnableCmd ( MCG\_Type \* *base*, bool *enable* ) [inline], [static]**

This function enables/disables the internal reference clock to use as the MCGIRCLK.

## Parameters

|               |                                                                                                                                                           |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | Base address for current MCG instance.                                                                                                                    |
| <i>enable</i> | Enable or disable internal reference clock. <ul style="list-style-type: none"> <li>• true: MCGIRCLK active</li> <li>• false: MCGIRCLK inactive</li> </ul> |

**45.2.8.16 static void CLOCK\_HAL\_SetInternalRefClkEnableInStopCmd ( MCG\_Type \* *base*, bool *enable* ) [inline], [static]**

This function controls whether or not the internal reference clock remains enabled when the MCG enters Stop mode.

## Parameters

|               |                                                                                                                                                                                                                                                                                                                                  |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | Base address for current MCG instance.                                                                                                                                                                                                                                                                                           |
| <i>enable</i> | 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> |

## MCG HAL driver

**45.2.8.17** `static void CLOCK_HAL_SetInternalRefClkMode ( MCG_Type * base,  
mcg_irc_mode_t mode ) [inline], [static]`

This function selects between the fast or slow internal reference clock source.

## Parameters

|               |                                                                                                                                                                                       |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | Base address for current MCG instance.                                                                                                                                                |
| <i>select</i> | Internal reference clock source. <ul style="list-style-type: none"> <li>• 0: Slow internal reference clock selected.</li> <li>• 1: Fast internal reference clock selected.</li> </ul> |

**45.2.8.18 static mcg\_irc\_mode\_t CLOCK\_HAL\_GetInternalRefClkMode ( MCG\_Type \* *base* ) [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>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

## Returns

## 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).

**45.2.8.19 void CLOCK\_HAL\_UpdateFastClkInternalRefDiv ( MCG\_Type \* *base*, uint8\_t *fcrdiv* )**

This function sets FCRDIV to a new value. FCRDIV cannot be changed when fast internal reference is enabled. If it is enabled, disable it , then set FCRDIV, and finally re enable it.

## Parameters

|               |                                               |
|---------------|-----------------------------------------------|
| <i>base</i>   | Base address for current MCG instance.        |
| <i>fcrdiv</i> | Fast Clock Internal Reference Divider Setting |

## MCG HAL driver

**45.2.8.20** void **CLOCK\_HAL\_SetOsc0Mode** ( MCG\_Type \* *base*, osc\_range\_t *range*,  
osc\_gain\_t *hgo*, osc\_src\_t *erefs* )

This function sets the OSC0 work mode, include frequency range select, high gain oscillator select, and external reference select.



## Parameters

|              |                                        |
|--------------|----------------------------------------|
| <i>base</i>  | Base address for current MCG instance. |
| <i>range</i> | Frequency range select.                |
| <i>hgo</i>   | High gain oscillator select.           |
| <i>erefs</i> | External reference select.             |

**45.2.8.21 static bool CLOCK\_HAL\_IsOsc0Stable ( MCG\_Type \* *base* ) [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>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

## Returns

True if OSC0 is stable.

**45.2.8.22 void CLOCK\_HAL\_EnableOsc0Monitor ( MCG\_Type \* *base*, mcg\_osc\_monitor\_mode\_t *mode* )**

This function enables the loss of clock monitoring circuit for the OSC0 external reference mux select. The monitor mode determines whether an interrupt or a reset request is generated following a loss of the OSC0 indication. External clock monitor should only be enabled when the MCG is in an operational mode that uses the external clock (FEE, FBE, PEE, PBE, or BLPE). Whenever the monitor is enabled, the value of the RANGE0 bits in the C2 register should not be changed. External clock monitor should be disabled before the MCG enters any Stop mode. Otherwise, a reset request may occur while in Stop mode. External clock monitor should also be disabled before entering VLPR or VLPW power modes if the MCG is in BLPE mode.

## Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

## MCG HAL driver

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>mode</i> | Generate interrupt or reset when OSC loss detected. |
|-------------|-----------------------------------------------------|

### 45.2.8.23 **static void** CLOCK\_HAL\_DisableOsc0Monitor ( MCG\_Type \* *base* ) [inline], [static]

This function disables the loss of clock monitoring circuit for the OSC0 external reference mux select.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

### 45.2.8.24 **static bool** CLOCK\_HAL\_IsOsc0MonitorEnabled ( MCG\_Type \* *base* ) [inline], [static]

This function checks whether the loss of clock monitoring circuit for the OSC0 external reference mux select is enabled or not.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

Returns

True if monitor is enabled.

### 45.2.8.25 **static bool** CLOCK\_HAL\_IsOsc0LostLock ( MCG\_Type \* *base* ) [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.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

Returns

True if loss of OSC0 has occurred.

**45.2.8.26** `static void CLOCK_HAL_ClearOsc0LostLock ( MCG_Type * base ) [inline],  
[static]`

This function clears the OSC0 Loss of Clock Status.

## MCG HAL driver

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

#### 45.2.8.27 `mcg_atm_error_t CLOCK_HAL_TrimInternalRefClk ( MCG_Type * base, 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 successful, it returns the `kMcgAtmErrorNone` and the frequency after trimming is received in the parameter `actualFreq`. If an error occurs, the error code is returned.

### Parameters

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

#### 45.2.8.28 `static bool CLOCK_HAL_IsAutoTrimMachineFailed ( MCG_Type * base ) [inline], [static]`

This function gets the Automatic Trim machine Fail Flag. 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 a Stop mode. Writing to `ATMF` clears the flag.

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

### Returns

True if ATM failed.

**45.2.8.29** `static void CLOCK_HAL_ClearAutoTrimMachineFailed ( MCG_Type * base )`  
`[inline], [static]`

This function clears the Automatic Trim machine Fail Flag.

## MCG HAL driver

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

### 45.2.8.30 `mcg_modes_t` `CLOCK_HAL_GetMcgMode ( MCG_Type * base )`

This function checks the MCG registers and determine current MCG mode.

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current MCG instance. |
|-------------|----------------------------------------|

### Returns

Current MCG mode or error code `mcg_modes_t`

### 45.2.8.31 `mcg_mode_error_t` `CLOCK_HAL_SetFeiMode ( MCG_Type * base, mcg_dco_range_select_t drs, void(*)(void) fllStableDelay, uint32_t * outClkFreq )`

This function sets MCG to FEI mode.

### Parameters

|                       |                                            |
|-----------------------|--------------------------------------------|
| <i>base</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

Error code

### 45.2.8.32 `mcg_mode_error_t` `CLOCK_HAL_SetFeeMode ( MCG_Type * base, mcg_oscsel_select_t oscselVal, uint8_t frdivVal, mcg_dm32_select_t dm32, mcg_dco_range_select_t drs, void(*)(void) fllStableDelay, uint32_t * outClkFreq )`

This function sets MCG to FEE mode.

## Parameters

|                      |                                            |
|----------------------|--------------------------------------------|
| <i>base</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>flStableDelay</i> | Delay function to make sure FLL is stable. |
| <i>outClkFreq</i>    | MCGCLKOUT frequency in new mode.           |

## Returns

Error code

**45.2.8.33** `mcg_mode_error_t CLOCK_HAL_SetFbiMode ( MCG_Type * base, mcg_dco_range_select_t drs, mcg_irc_mode_t ircSelect, uint8_t frdivVal, void(*)(void) flStableDelay, uint32_t * outClkFreq )`

This function sets MCG to FBI mode.

## Parameters

|                      |                                            |
|----------------------|--------------------------------------------|
| <i>base</i>          | Base address for current MCG instance.     |
| <i>drs</i>           | The DCO range selection.                   |
| <i>ircselect</i>     | The internal reference clock to select.    |
| <i>flStableDelay</i> | Delay function to make sure FLL is stable. |
| <i>outClkFreq</i>    | MCGCLKOUT frequency in new mode.           |

## Returns

Error code

**45.2.8.34** `mcg_mode_error_t CLOCK_HAL_SetFbeMode ( MCG_Type * base, mcg_oscsel_select_t oscselVal, uint8_t frdivVal, mcg_dmx32_select_t dmx32, mcg_dco_range_select_t drs, void(*)(void) flStableDelay, uint32_t * outClkFreq )`

This function sets MCG to FBE mode.

## MCG HAL driver

### Parameters

|                       |                                            |
|-----------------------|--------------------------------------------|
| <i>base</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

Error code

#### 45.2.8.35 `mcg_mode_error_t` `CLOCK_HAL_SetBlpiMode ( MCG_Type * base, uint8_t fcrdivVal, mcg_irc_mode_t ircSelect, uint32_t * outClkFreq )`

This function sets MCG to BLPI mode.

### Parameters

|                   |                                         |
|-------------------|-----------------------------------------|
| <i>base</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

Error code

#### 45.2.8.36 `mcg_mode_error_t` `CLOCK_HAL_SetBlpeMode ( MCG_Type * base, mcg_oscsel_select_t oscselVal, uint32_t * outClkFreq )`

This function sets MCG to BLPE mode.

### Parameters

\_\_\_\_\_



|                   |                                        |
|-------------------|----------------------------------------|
| <i>base</i>       | Base address for current MCG instance. |
| <i>oscselval</i>  | OSCSEL in FEE mode.                    |
| <i>outClkFreq</i> | MCGCLKOUT frequency in new mode.       |

Returns

Error code

**45.2.8.37** `mcg_mode_error_t CLOCK_HAL_SetPbeMode ( MCG_Type * base, 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>base</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

Error code

**45.2.8.38** `mcg_mode_error_t CLOCK_HAL_SetPeeMode ( MCG_Type * base, uint32_t * outClkFreq )`

This function sets MCG to PBE mode.

Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <i>base</i>       | Base address for current MCG instance. |
| <i>outClkFreq</i> | MCGCLKOUT frequency in new mode.       |

Returns

Error code

---

## MCG HAL driver

### 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 46

# Multipurpose Clock Generator Lite (MCG\_Lite)

### 46.1 Overview

The Kinetis SDK provides the HAL drivers for the Multipurpose Clock Generator Lite block of Kinetis devices.

### Modules

- [MCG\\_Lite HAL driver](#)

## MCG\_Lite HAL driver

### 46.2 MCG\_Lite HAL driver

#### 46.2.1 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.

#### 46.2.2 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 frequency
frequency = CLOCK_HAL_GetOutClk();
```

#### 46.2.3 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](#) {  
  [kMcgliteClkSrcHirc](#),  
  [kMcgliteClkSrcLirc](#),  
  [kMcgliteClkSrcExt](#) }  
*MCG\_Lite clock source selection.*

- enum mcglite\_lirc\_select\_t {  
kMcgliteLircSel2M,  
kMcgliteLircSel8M }  
*MCG\_Lite LIRC select.*
- enum mcglite\_lirc\_div\_t {  
kMcgliteLircDivBy1 = 0U,  
kMcgliteLircDivBy2,  
kMcgliteLircDivBy4,  
kMcgliteLircDivBy8,  
kMcgliteLircDivBy16,  
kMcgliteLircDivBy32,  
kMcgliteLircDivBy64,  
kMcgliteLircDivBy128 }  
*MCG\_Lite divider factor selection for clock source.*
- enum osc\_src\_t {  
kOscSrcExt,  
kOscSrcOsc,  
kOscSrcExt,  
kOscSrcOsc }  
*MCG\_Lite external clock Select.*
- enum osc\_range\_t {  
kOscRangeLow,  
kOscRangeHigh,  
kOscRangeVeryHigh,  
kOscRangeVeryHigh1,  
kOscRangeLow,  
kOscRangeHigh,  
kOscRangeVeryHigh,  
kOscRangeVeryHigh1 }  
*MCG frequency range select.*
- enum osc\_gain\_t {  
kOscGainLow,  
kOscGainHigh,  
kOscGainLow,  
kOscGainHigh }  
*MCG high gain oscillator select.*
- enum mcglite\_mode\_t {  
kMcgliteModeHirc48M,  
kMcgliteModeLirc8M,  
kMcgliteModeLirc2M,  
kMcgliteModeExt,  
kMcgliteModeStop,  
kMcgliteModeError }  
*MCG\_Lite clock mode definitions.*
- enum mcglite\_mode\_error\_t {  
kMcgliteModeErrNone = 0x00,

## MCG\_Lite HAL driver

`kMcgliteModeErrExt = 0x01 }`

*MCG\_Lite mode transition API error code definitions.*

## MCG\_Lite output clock access API

- `uint32_t CLOCK_HAL_GetPeripheralClk` (MCG\_Type \*base)  
*Gets the current MCGPCLK frequency.*
- `uint32_t CLOCK_HAL_GetLircClk` (MCG\_Type \*base)  
*Gets the current MCG\_Lite low internal reference clock(2MHz or 8MHz)*
- `uint32_t CLOCK_HAL_GetLircDiv1Clk` (MCG\_Type \*base)  
*Gets the current MCG\_Lite LIRC\_DIV1\_CLK frequency.*
- `uint32_t CLOCK_HAL_GetInternalRefClk` (MCG\_Type \*base)  
*Gets the current MCGIRCLK frequency.*
- `uint32_t CLOCK_HAL_GetOutClk` (MCG\_Type \*base)  
*Gets the current MCGOUTCLK frequency.*

## MCG\_Lite control register access API

- static void `CLOCK_HAL_SetLircSelMode` (MCG\_Type \*base, `mcglite_lirc_select_t` select)  
*Sets the Low Internal Reference Select.*
- static void `CLOCK_HAL_SetLircRefDiv` (MCG\_Type \*base, `mcglite_lirc_div_t` setting)  
*Sets the low internal reference divider 1.*
- static void `CLOCK_HAL_SetLircDiv2` (MCG\_Type \*base, `mcglite_lirc_div_t` setting)  
*Sets the low internal reference divider 2.*
- static void `CLOCK_HAL_SetLircCmd` (MCG\_Type \*base, bool enable)  
*Enables the Low Internal Reference Clock setting.*
- static void `CLOCK_HAL_SetLircStopCmd` (MCG\_Type \*base, bool enable)  
*Sets the Low Internal Reference Clock disabled or not in STOP mode.*
- static void `CLOCK_HAL_SetHircCmd` (MCG\_Type \*base, bool enable)  
*Enable or disable the High Internal Reference Clock setting.*
- static void `CLOCK_HAL_SetExtRefSelMode0` (MCG\_Type \*base, `osc_src_t` select)  
*Sets the External Reference Select.*
- static `mcglite_mcgoutclk_source_t` `CLOCK_HAL_GetClkSrcStat` (MCG\_Type \*base)  
*Gets the Clock Mode Status.*
- static bool `CLOCK_HAL_IsOscStable` (MCG\_Type \*base)  
*Gets the OSC Initialization Status.*

## MCG\_Lite clock mode API

- `mcglite_mode_t` `CLOCK_HAL_GetMode` (MCG\_Type \*base)  
*Gets the current MCG\_Lite clock mode.*
- `mcglite_mode_error_t` `CLOCK_HAL_SetHircMode` (MCG\_Type \*base, `uint32_t` \*outClkFreq)  
*Sets the MCG\_Lite to HIRC mode.*
- `mcglite_mode_error_t` `CLOCK_HAL_SetLircMode` (MCG\_Type \*base, `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` (MCG\_Type \*base, `uint32_t` \*outClkFreq)

Sets the MCG\_Lite to EXT mode.

## 46.2.4 Enumeration Type Documentation

### 46.2.4.1 enum \_mcglite\_constant

#### 46.2.4.2 enum mcglite\_mcgoutclk\_source\_t

Enumerator

- kMcgliteClkSrcHirc* MCGOUTCLK source is HIRC.
- kMcgliteClkSrcLirc* MCGOUTCLK source is LIRC.
- kMcgliteClkSrcExt* MCGOUTCLK source is external clock source.

#### 46.2.4.3 enum mcglite\_lirc\_select\_t

Enumerator

- kMcgliteLircSel2M* slow internal reference(LIRC) 2MHz clock selected
- kMcgliteLircSel8M* slow internal reference(LIRC) 8MHz clock selected

#### 46.2.4.4 enum mcglite\_lirc\_div\_t

Enumerator

- kMcgliteLircDivBy1* divider is 1
- kMcgliteLircDivBy2* divider is 2
- kMcgliteLircDivBy4* divider is 4
- kMcgliteLircDivBy8* divider is 8
- kMcgliteLircDivBy16* divider is 16
- kMcgliteLircDivBy32* divider is 32
- kMcgliteLircDivBy64* divider is 64
- kMcgliteLircDivBy128* divider is 128

#### 46.2.4.5 enum osc\_src\_t

Enumerator

- kOscSrcExt* External reference clock requested.
- kOscSrcOsc* Oscillator requested.
- kOscSrcExt* Selects external input clock.
- kOscSrcOsc* Selects Oscillator.

## MCG\_Lite HAL driver

### 46.2.4.6 enum osc\_range\_t

Enumerator

*kOscRangeLow* Low frequency range selected for the crystal OSC.  
*kOscRangeHigh* High frequency range selected for the crystal OSC.  
*kOscRangeVeryHigh* Very High frequency range selected for the crystal OSC.  
*kOscRangeVeryHigh1* Very High frequency range selected for the crystal OSC.  
*kOscRangeLow* Low frequency range selected for the crystal OSC.  
*kOscRangeHigh* High frequency range selected for the crystal OSC.  
*kOscRangeVeryHigh* Very High frequency range selected for the crystal OSC.  
*kOscRangeVeryHigh1* Very High frequency range selected for the crystal OSC.

### 46.2.4.7 enum osc\_gain\_t

Enumerator

*kOscGainLow* Configure crystal oscillator for low-power operation.  
*kOscGainHigh* Configure crystal oscillator for high-gain operation.  
*kOscGainLow* Configure crystal oscillator for low-power operation.  
*kOscGainHigh* Configure crystal oscillator for high-gain operation.

### 46.2.4.8 enum mcglite\_mode\_t

Enumerator

*kMcgliteModeHirc48M* clock mode is HIRC 48M  
*kMcgliteModeLirc8M* clock mode is LIRC 8M  
*kMcgliteModeLirc2M* clock mode is LIRC 2M  
*kMcgliteModeExt* clock mode is EXT  
*kMcgliteModeStop* clock mode is STOP  
*kMcgliteModeError* Unknown mode.

### 46.2.4.9 enum mcglite\_mode\_error\_t

Enumerator

*kMcgliteModeErrNone* • No error  
*kMcgliteModeErrExt* • External clock source not available.



## 46.2.5 Function Documentation

### 46.2.5.1 uint32\_t CLOCK\_HAL\_GetPeripheralClk ( MCG\_Type \* *base* )

This function returns the MCGPCLK 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>base</i> | MCG_Lite register base address. |
|-------------|---------------------------------|

### Returns

Frequency value in Hertz of MCGPCLK.

#### 46.2.5.2 uint32\_t CLOCK\_HAL\_GetLircClk ( MCG\_Type \* *base* )

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>base</i> | MCG_Lite register base address. |
|-------------|---------------------------------|

### Returns

Frequency value in Hertz of the MCG\_Lite LIRC.

#### 46.2.5.3 uint32\_t CLOCK\_HAL\_GetLircDiv1Clk ( MCG\_Type \* *base* )

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>base</i> | MCG_Lite register base address. |
|-------------|---------------------------------|

### Returns

Frequency value in Hertz of the MCG\_Lite LIRC\_DIV1\_CLK.

#### 46.2.5.4 uint32\_t CLOCK\_HAL\_GetInternalRefClk ( MCG\_Type \* *base* )

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>base</i> | MCG_Lite register base address. |
|-------------|---------------------------------|

Returns

Frequency value in Hertz of MCGIRCLK.

**46.2.5.5 uint32\_t CLOCK\_HAL\_GetOutClk ( MCG\_Type \* *base* )**

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.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | MCG_Lite register base address. |
|-------------|---------------------------------|

Returns

Frequency value in Hertz of MCGOUTCLK.

**46.2.5.6 static void CLOCK\_HAL\_SetLircSelMode ( MCG\_Type \* *base*, mcglite\_lirc\_select\_t *select* ) [inline], [static]**

This function sets the LIRC to work at 2MHz or 8MHz.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | MCG_Lite register base address. |
| <i>select</i> | 2MHz or 8MHz.                   |

**46.2.5.7 static void CLOCK\_HAL\_SetLircRefDiv ( MCG\_Type \* *base*, 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>base</i>    | MCG_Lite register base address. |
| <i>setting</i> | LIRC divider 1 setting value.   |

### 46.2.5.8 static void CLOCK\_HAL\_SetLircDiv2 ( MCG\_Type \* *base*, mcglite\_lirc\_div\_t *setting* ) [inline], [static]

This function sets the low internal reference divider 2.

Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>base</i>    | MCG_Lite register base address. |
| <i>setting</i> | LIRC divider 2 setting value.   |

### 46.2.5.9 static void CLOCK\_HAL\_SetLircCmd ( MCG\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables/disables the low internal reference clock.

Parameters

|               |                                                                                                                                                                               |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |

### 46.2.5.10 static void CLOCK\_HAL\_SetLircStopCmd ( MCG\_Type \* *base*, 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>base</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> |

```
46.2.5.11 static void CLOCK_HAL_SetHircCmd (MCG_Type * base, bool enable)
 [inline], [static]
```

This function enables/disables the internal reference clock for use as MCGPCLK.

## MCG\_Lite HAL driver

### Parameters

|               |                                                                                                                                                 |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |

### 46.2.5.12 static void CLOCK\_HAL\_SetExtRefSelMode0 ( MCG\_Type \* *base*, osc\_src\_t *select* ) [inline], [static]

This function selects the source for the external reference clock. Refer to the Oscillator (OSC) for more details.

### Parameters

|               |                                                                                                                                                  |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |

### 46.2.5.13 static mcglite\_mcgoutclk\_source\_t CLOCK\_HAL\_GetClkSrcStat ( MCG\_Type \* *base* ) [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>base</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.

#### 46.2.5.14 `static bool CLOCK_HAL_IsOscStable ( MCG_Type * base ) [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>base</i> | MCG_Lite register base address. |
|-------------|---------------------------------|

Returns

OSC initialization status

#### 46.2.5.15 `mcglite_mode_t CLOCK_HAL_GetMode ( MCG_Type * base )`

This is an internal function that checks the MCG registers and determine the current MCG\_lite mode.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | MCG_Lite register base address. |
|-------------|---------------------------------|

Returns

Current MCG\_Lite mode or error code.

#### 46.2.5.16 `mcglite_mode_error_t CLOCK_HAL_SetHircMode ( MCG_Type * base, uint32_t * outClkFreq )`

This is an internal function that changes MCG\_Lite to HRIC mode.

Parameters

|                   |                                  |
|-------------------|----------------------------------|
| <i>base</i>       | MCG_Lite register base address.  |
| <i>outclkfreq</i> | MCGOUTCLK frequency in new mode. |

Returns

Error code.

#### 46.2.5.17 `mcglite_mode_error_t CLOCK_HAL_SetLircMode ( MCG_Type * base, 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.

## MCG\_Lite HAL driver

### Parameters

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

#### 46.2.5.18 mcglite\_mode\_error\_t CLOCK\_HAL\_SetExtMode ( MCG\_Type \* *base*, 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.

### Parameters

|                   |                                  |
|-------------------|----------------------------------|
| <i>base</i>       | MCG_Lite register base address.  |
| <i>outclkfreq</i> | MCGOUTCLK frequency in new mode. |

### Returns

Error code.



---

## Chapter 47

# Memory-Mapped Divide and Square Root(MMDVSQ)

### 47.1 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

### 47.2 MMDVSQ HAL driver

#### 47.2.1 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.

#### 47.2.2 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](#) (MMDVSQ\_Type \*base, uint32\_t dividend, uint32\_t divisor)  
*perform the current MMDVSQ unsigned divide operation and get remainder*
- uint32\_t [MMDVSQ\\_HAL\\_DivUQ](#) (MMDVSQ\_Type \*base, uint32\_t dividend, uint32\_t divisor)  
*perform the current MMDVSQ unsigned divide operation and get quotient*
- uint32\_t [MMDVSQ\\_HAL\\_DivSR](#) (MMDVSQ\_Type \*base, uint32\_t dividend, uint32\_t divisor)  
*perform the current MMDVSQ signed divide operation and get remainder*
- uint32\_t [MMDVSQ\\_HAL\\_DivSQ](#) (MMDVSQ\_Type \*base, uint32\_t dividend, uint32\_t divisor)  
*perform the current MMDVSQ signed divide operation and get quotient*
- uint16\_t [MMDVSQ\\_HAL\\_Sqrt](#) (MMDVSQ\_Type \*base, uint32\_t radicand)  
*set the current MMDVSQ square root operation*

## MMDVSQ control register access API

- static [mmdvsq\\_execution\\_status\\_t MMDVSQ\\_HAL\\_GetExecutionStatus](#) (MMDVSQ\_Type \*base)  
*Get the current MMDVSQ execution status.*
- static bool [MMDVSQ\\_HAL\\_GetBusyStatus](#) (MMDVSQ\_Type \*base)  
*Get the current MMDVSQ BUSY status.*
- static void [MMDVSQ\\_HAL\\_SetDivideFastStart](#) (MMDVSQ\_Type \*base, bool enable)  
*set the current MMDVSQ divide fast start*
- static bool [MMDVSQ\\_HAL\\_GetDivideFastStart](#) (MMDVSQ\_Type \*base)  
*get the current MMDVSQ divide fast start setting*
- static void [MMDVSQ\\_HAL\\_SetDivideByZero](#) (MMDVSQ\_Type \*base, bool enable)  
*set the current MMDVSQ divide by zero detection*
- static bool [MMDVSQ\\_HAL\\_GetDivideByZeroSetting](#) (MMDVSQ\_Type \*base)  
*get the current MMDVSQ divide by zero setting*
- static bool [MMDVSQ\\_HAL\\_GetDivideByZeroStatus](#) (MMDVSQ\_Type \*base)  
*get the current MMDVSQ divide by zero status*
- static void [MMDVSQ\\_HAL\\_SetRemainderCalculation](#) (MMDVSQ\_Type \*base, bool enable)  
*set the current MMDVSQ divide remainder calculation*
- static bool [MMDVSQ\\_HAL\\_GetRemainderCalculation](#) (MMDVSQ\_Type \*base)  
*get the current MMDVSQ divide remainder calculation*
- static void [MMDVSQ\\_HAL\\_SetUnsignedCalculation](#) (MMDVSQ\_Type \*base, bool enable)  
*set the current MMDVSQ unsigned divide calculation*
- static bool [MMDVSQ\\_HAL\\_GetUnsignedCalculation](#) (MMDVSQ\_Type \*base)  
*get the current MMDVSQ unsigned divide calculation*
- static uint32\_t [MMDVSQ\\_HAL\\_GetResult](#) (MMDVSQ\_Type \*base)  
*get the current MMDVSQ operation result*
- static void [MMDVSQ\\_HAL\\_SetDividend](#) (MMDVSQ\_Type \*base, uint32\_t dividend)  
*set the current MMDVSQ dividend value*
- static uint32\_t [MMDVSQ\\_HAL\\_GetDividend](#) (MMDVSQ\_Type \*base)  
*get the current MMDVSQ dividend value*
- static void [MMDVSQ\\_HAL\\_SetDivisor](#) (MMDVSQ\_Type \*base, uint32\_t divisor)  
*set the current MMDVSQ divisor value*
- static uint32\_t [MMDVSQ\\_HAL\\_GetDivisor](#) (MMDVSQ\_Type \*base)  
*get the current MMDVSQ divisor value*
- static void [MMDVSQ\\_HAL\\_SetRadicand](#) (MMDVSQ\_Type \*base, uint32\_t radicand)  
*set the current MMDVSQ radicand value*

## MMDVSQ HAL driver

### 47.2.3 Function Documentation

#### 47.2.3.1 `uint32_t MMDVSQ_HAL_DivUR ( MMDVSQ_Type * base, 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>base</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.

**47.2.3.2 uint32\_t MMDVSQ\_HAL\_DivUQ ( MMDVSQ\_Type \* *base*, 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>base</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.

**47.2.3.3 uint32\_t MMDVSQ\_HAL\_DivSR ( MMDVSQ\_Type \* *base*, 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>base</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.

### 47.2.3.4 `uint32_t MMDVSQ_HAL_DivSQ ( MMDVSQ_Type * base, 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>base</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.

### 47.2.3.5 `uint16_t MMDVSQ_HAL_Sqrt ( MMDVSQ_Type * base, uint32_t radicand )`

This function performs the MMDVSQ square root operation and return the sqrt result of given radicand-value. It is in block mode. For non-block mode, other HAL routines can be used.

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>base</i>     | Base address for current MMDVSQ instance. |
| <i>radicand</i> | - Radicand value                          |

Returns

Square root calculation result in the MMDVSQ\_RES register.

### 47.2.3.6 `static mmdvsq_execution_status_t MMDVSQ_HAL_GetExecutionStatus ( MMDVSQ_Type * base ) [inline], [static]`

This function checks the current MMDVSQ execution status

Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | Base address for current MMDVSQ instance. |
|-------------|-------------------------------------------|

Returns

Current MMDVSQ execution status

**47.2.3.7 static bool MMDVSQ\_HAL\_GetBusyStatus ( MMDVSQ\_Type \* *base* )  
[inline], [static]**

This function checks the current MMDVSQ BUSY status

Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | Base address for current MMDVSQ instance. |
|-------------|-------------------------------------------|

Returns

MMDVSQ is busy or idle

**47.2.3.8 static void MMDVSQ\_HAL\_SetDivideFastStart ( MMDVSQ\_Type \* *base*, bool  
*enable* ) [inline], [static]**

This function sets the MMDVSQ divide fast start.

Parameters

|               |                                                                                                                                                                                              |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |

**47.2.3.9 static bool MMDVSQ\_HAL\_GetDivideFastStart ( MMDVSQ\_Type \* *base* )  
[inline], [static]**

This function gets the MMDVSQ divide fast start setting

## MMDVSQ HAL driver

### Parameters

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

#### 47.2.3.10 `static void MMDVSQ_HAL_SetDivdeByZero ( MMDVSQ_Type * base, bool enable ) [inline], [static]`

This function sets the MMDVSQ divide by zero detection

### Parameters

|               |                                                                                                                                                                                   |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |

#### 47.2.3.11 `static bool MMDVSQ_HAL_GetDivdeByZeroSetting ( MMDVSQ_Type * base ) [inline], [static]`

This function gets the MMDVSQ divide by zero setting

### Parameters

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

#### 47.2.3.12 `static bool MMDVSQ_HAL_GetDivdeByZeroStatus ( MMDVSQ_Type * base ) [inline], [static]`

This function gets the MMDVSQ divide by zero status



Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | Base address for current MMDVSQ instance. |
|-------------|-------------------------------------------|

Returns

MMDVSQ divide is non-zero divisor or zero divisor

- true: zero divisor.
  - false: non-zero divisor.

**47.2.3.13 static void MMDVSQ\_HAL\_SetRemainderCalculation ( MMDVSQ\_Type \* *base*, bool *enable* ) [inline], [static]**

This function sets the MMDVSQ divide remainder calculation

Parameters

|               |                                                                                                                                                                                                     |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |

**47.2.3.14 static bool MMDVSQ\_HAL\_GetRemainderCalculation ( MMDVSQ\_Type \* *base* ) [inline], [static]**

This function gets the MMDVSQ divide remainder calculation

Parameters

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

**47.2.3.15 static void MMDVSQ\_HAL\_SetUnsignedCalculation ( MMDVSQ\_Type \* *base*, bool *enable* ) [inline], [static]**

This function sets the MMDVSQ unsigned divide calculation

## MMDVSQ HAL driver

### Parameters

|               |                                                                                                                                                                                                     |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |

### 47.2.3.16 static bool MMDVSQ\_HAL\_GetUnsignedCalculation ( MMDVSQ\_Type \* *base* ) [inline], [static]

This function gets the MMDVSQ unsigned divide calculation

### Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | Base address for current MMDVSQ instance. |
|-------------|-------------------------------------------|

### Returns

MMDVSQ divide is unsigned divide operation

- true: perform an unsigned divide.
- false: perform a signed divide

### 47.2.3.17 static uint32\_t MMDVSQ\_HAL\_GetResult ( MMDVSQ\_Type \* *base* ) [inline], [static]

This function gets the MMDVSQ operation result

### Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | Base address for current MMDVSQ instance. |
|-------------|-------------------------------------------|

### Returns

MMDVSQ operation result

### 47.2.3.18 static void MMDVSQ\_HAL\_SetDividend ( MMDVSQ\_Type \* *base*, uint32\_t *dividend* ) [inline], [static]

This function sets the MMDVSQ dividend value

Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>base</i>     | Base address for current MMDVSQ instance. |
| <i>dividend</i> | Dividend value for divide calculations.   |

**47.2.3.19 static uint32\_t MMDVSQ\_HAL\_GetDividend ( MMDVSQ\_Type \* *base* )  
[inline], [static]**

This function gets the MMDVSQ dividend value

Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | Base address for current MMDVSQ instance. |
|-------------|-------------------------------------------|

Returns

MMDVSQ dividend value

**47.2.3.20 static void MMDVSQ\_HAL\_SetDivisor ( MMDVSQ\_Type \* *base*, uint32\_t *divisor* )  
[inline], [static]**

This function sets the MMDVSQ divisor value

Parameters

|                |                                           |
|----------------|-------------------------------------------|
| <i>base</i>    | Base address for current MMDVSQ instance. |
| <i>divisor</i> | Divisor value for divide calculations..   |

**47.2.3.21 static uint32\_t MMDVSQ\_HAL\_GetDivisor ( MMDVSQ\_Type \* *base* )  
[inline], [static]**

This function gets the MMDVSQ divisor value

Parameters

|             |                                           |
|-------------|-------------------------------------------|
| <i>base</i> | Base address for current MMDVSQ instance. |
|-------------|-------------------------------------------|

Returns

MMDVSQ divisor value

## MMDVSQ HAL driver

**47.2.3.22** `static void MMDVSQ_HAL_SetRadicand ( MMDVSQ_Type * base, uint32_t radicand ) [inline], [static]`

This function sets the MMDVSQ radicand value

## Parameters

|                 |                                           |
|-----------------|-------------------------------------------|
| <i>base</i>     | Base address for current MMDVSQ instance. |
| <i>radicand</i> | Radicand value of Sqrt.                   |



## Chapter 48 Oscillator (OSC)

### 48.1 Overview

The Kinetis SDK provides a HAL driver for the Oscillator (OSC) block of Kinetis devices.

#### Modules

- [OSC HAL driver](#)

## OSC HAL driver

### 48.2 OSC HAL driver

#### 48.2.1 Overview

The chapter 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 driver provides APIs to:

- Configure OSCERCLK.
- Configure the capacitor load.

#### 48.2.2 OSCERCLK Configure APIs

To enable/disable OSCERCLK, use the functions [OSC\\_HAL\\_SetExternalRefClkCmd\(\)](#) and [OSC\\_HAL\\_SetExternalRefClkInStopModeCmd\(\)](#). These functions configure the OSCERCLK enabling or disabling in normal mode or stop mode.

The function [OSC\\_HAL\\_SetExternalRefClkDiv\(\)](#) configures the OSCERCLK divider.

#### 48.2.3 Capacitor Load Configure APIs

To configure capacitor load, use the function [OSC\\_HAL\\_SetCapacitor](#) and pass the capacitor load as parameter. Example:

```
#include "fsl_osc_hal.h"

// To enable only 2 pF and 8 pF capacitor load, do this:
OSC_HAL_SetCapacitor(OSC, kOscCapacitor2p |
 kOscCapacitor8p);
```

#### 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

- static void [OSC\\_HAL\\_SetExternalRefClkCmd](#) (OSC\_Type \*base, bool enable)  
*Enables the external reference clock for the oscillator.*
- static bool [OSC\\_HAL\\_GetExternalRefClkCmd](#) (OSC\_Type \*base)  
*Gets the external reference clock enable setting for the oscillator.*
- static void [OSC\\_HAL\\_SetExternalRefClkInStopModeCmd](#) (OSC\_Type \*base, bool enable)  
*Enables/disables the external reference clock in stop mode.*
- void [OSC\\_HAL\\_SetCapacitor](#) (OSC\_Type \*base, uint32\_t bitMask)  
*Sets the capacitor configuration for the oscillator.*

## 48.2.4 Enumeration Type Documentation

### 48.2.4.1 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

## 48.2.5 Function Documentation

### 48.2.5.1 static void OSC\_HAL\_SetExternalRefClkCmd ( OSC\_Type \* *base*, bool *enable* ) [inline], [static]

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>base</i>   | Oscillator register base address |
| <i>enable</i> | enable/disable the clock         |

### 48.2.5.2 static bool OSC\_HAL\_GetExternalRefClkCmd ( OSC\_Type \* *base* ) [inline], [static]

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>base</i> | Oscillator register base address |
|-------------|----------------------------------|

Returns

Clock enable/disable setting

### 48.2.5.3 static void OSC\_HAL\_SetExternalRefClkInStopModeCmd ( OSC\_Type \* *base*, bool *enable* ) [inline], [static]

This function enables/disables the external reference clock (OSCERCLK) when an MCU enters the stop mode.

Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | Oscillator register base address |
| <i>enable</i> | enable/disable setting           |

### 48.2.5.4 void OSC\_HAL\_SetCapacitor ( OSC\_Type \* *base*, uint32\_t *bitMask* )

This function sets 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>base</i>    | Oscillator register base address        |
| <i>bitMask</i> | Bit mask for the capacitor load option. |

Example:

```
// To enable only 2 pF and 8 pF capacitor load, please use like this.
OSC_HAL_SetCapacitor(OSC, kOscCapacitor2p |
 kOscCapacitor8p);
```

---

## Chapter 49 Power Management Controller (PMC)

### 49.1 Overview

The Kinetis SDK provides a HAL driver for the Power Management Controller (PMC) block of Kinetis devices.

#### Modules

- [PMC HAL driver](#)

## PMC HAL driver

### 49.2 PMC HAL driver

#### 49.2.1 Overview

The Power Management Controller (PMC) contains the internal voltage regulator, power on reset (POR), and low voltage detect system.

The PMC HAL driver provides these APIs:

- APIs for low voltage detection
- APIs for band gap setting
- APIs for I/O latch state

The low voltage detection can be configured by these functions: [PMC\\_HAL\\_LowVoltDetectConfig\(\)](#) and [PMC\\_HAL\\_LowVoltWarnConfig\(\)](#).

Use the function [PMC\\_HAL\\_GetLowVoltDetectFlag\(\)/PMC\\_HAL\\_GetLowVoltWarnFlag\(\)](#) to check the low voltage detection state. Use the function [PMC\\_HAL\\_SetLowVoltDetectAck\(\)/PMC\\_HAL\\_SetLowVoltWarnAck\(\)](#) to clear the flags.

To configure the band gap, use the [PMC\\_HAL\\_BandgapBufferConfig\(\)](#) function. For I/O latch state, use the [PMC\\_HAL\\_GetAckIsolation\(\)/PMC\\_HAL\\_ClearAckIsolation\(\)](#) function.

#### Files

- file [fsl\\_pmc\\_hal.h](#)

#### Data Structures

- struct [pmc\\_bandgap\\_buffer\\_config\\_t](#)  
*Bandgap Buffer configuration. [More...](#)*

#### 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.*

## Power Management Controller Control APIs

- static void [PMC\\_HAL\\_LowVoltDetectConfig](#) (PMC\_Type \*base, bool enableInt, bool enableReset, [pmc\\_low\\_volt\\_detect\\_volt\\_select\\_t](#) voltSelect)  
*Configure the low voltage detect setting.*
- static void [PMC\\_HAL\\_SetLowVoltDetectAck](#) (PMC\_Type \*base)  
*Low-Voltage Detect Acknowledge.*
- static bool [PMC\\_HAL\\_GetLowVoltDetectFlag](#) (PMC\_Type \*base)  
*Low-Voltage Detect Flag Read.*
- static void [PMC\\_HAL\\_LowVoltWarnConfig](#) (PMC\_Type \*base, bool enableInt, [pmc\\_low\\_volt\\_warn\\_volt\\_select\\_t](#) voltSelect)  
*Configure the low voltage warning setting.*
- static void [PMC\\_HAL\\_SetLowVoltWarnAck](#) (PMC\_Type \*base)  
*Low-Voltage Warning Acknowledge.*
- static bool [PMC\\_HAL\\_GetLowVoltWarnFlag](#) (PMC\_Type \*base)  
*Low-Voltage Warning Flag Read.*
- static void [PMC\\_HAL\\_BandgapBufferConfig](#) (PMC\_Type \*base, [pmc\\_bandgap\\_buffer\\_config\\_t](#) \*config)  
*Configures the PMC bandgap.*
- static uint8\_t [PMC\\_HAL\\_GetAckIsolation](#) (PMC\_Type \*base)  
*Gets the acknowledge isolation value.*
- static void [PMC\\_HAL\\_ClearAckIsolation](#) (PMC\_Type \*base)  
*Clears an acknowledge isolation.*
- static uint8\_t [PMC\\_HAL\\_GetRegulatorStatus](#) (PMC\_Type \*base)  
*Gets the Regulator regulation status.*

## 49.2.2 Data Structure Documentation

### 49.2.2.1 struct pmc\_bandgap\_buffer\_config\_t

#### Data Fields

- bool [enable](#)  
*Enable bandgap buffer.*

#### 49.2.2.1.0.73 Field Documentation

##### 49.2.2.1.0.73.1 bool pmc\_bandgap\_buffer\_config\_t::enable

## 49.2.3 Enumeration Type Documentation

### 49.2.3.1 enum pmc\_low\_volt\_warn\_volt\_select\_t

Enumerator

- kPmcLowVoltWarnVoltLowTrip*** Low trip point selected (VLVW = VLWV1)
- kPmcLowVoltWarnVoltMid1Trip*** Mid 1 trip point selected (VLVW = VLWV2)
- kPmcLowVoltWarnVoltMid2Trip*** Mid 2 trip point selected (VLVW = VLWV3)

## PMC HAL driver

*kPmcLowVoltWarnVoltHighTrip* High trip point selected (VLVW = VLVW4)

### 49.2.3.2 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 LVDP )

## 49.2.4 Function Documentation

**49.2.4.1 static void PMC\_HAL\_LowVoltDetectConfig ( PMC\_Type \* *base*, bool *enableInt*, bool *enableReset*, pmc\_low\_volt\_detect\_volt\_select\_t *voltSelect* ) [inline], [static]**

This function configures the low voltage detect setting, including the trip point voltage setting, enable interrupt or not, enable MCU reset or not.

Parameters

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <i>base</i>        | Base address for current PMC instance.           |
| <i>enableInt</i>   | Enable interrupt or not when low voltage detect. |
| <i>enableReset</i> | Enable MCU reset or not when low voltage detect. |
| <i>voltSelect</i>  | Low voltage detect trip point voltage.           |

**49.2.4.2 static void PMC\_HAL\_SetLowVoltDetectAck ( PMC\_Type \* *base* ) [inline], [static]**

This function acknowledges the low voltage detection errors (write 1 to clear LVDF).

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current PMC instance. |
|-------------|----------------------------------------|

**49.2.4.3 static bool PMC\_HAL\_GetLowVoltDetectFlag ( PMC\_Type \* *base* ) [inline], [static]**

This function reads the current LVDF status. If it returns 1, a low voltage event is detected.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current PMC instance. |
|-------------|----------------------------------------|

Returns

Current low voltage detect flag

- true: Low-Voltage detected
- false: Low-Voltage not detected

**49.2.4.4 static void PMC\_HAL\_LowVoltWarnConfig ( PMC\_Type \* *base*, bool *enableInt*, pmc\_low\_volt\_warn\_volt\_select\_t *voltSelect* ) [inline], [static]**

This function configures the low voltage warning setting, including the trip point voltage setting and enable interrupt or not.

Parameters

|                   |                                                  |
|-------------------|--------------------------------------------------|
| <i>base</i>       | Base address for current PMC instance.           |
| <i>enableInt</i>  | Enable interrupt or not when low voltage detect. |
| <i>voltSelect</i> | Low voltage detect trip point voltage.           |

**49.2.4.5 static void PMC\_HAL\_SetLowVoltWarnAck ( PMC\_Type \* *base* ) [inline], [static]**

This function acknowledges the low voltage warning errors (write 1 to clear LVWF).

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current PMC instance. |
|-------------|----------------------------------------|

**49.2.4.6 static bool PMC\_HAL\_GetLowVoltWarnFlag ( PMC\_Type \* *base* ) [inline], [static]**

This function polls the current LVWF status. When 1 is returned, it indicates a low-voltage warning event. LVWF is set when V Supply transitions below the trip point or after reset and V Supply is already below the V LVW.

## PMC HAL driver

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current PMC instance. |
|-------------|----------------------------------------|

### Returns

Current LVWF status

- true: Low-Voltage Warning Flag is set.
- false: the Low-Voltage Warning does not happen.

**49.2.4.7 static void PMC\_HAL\_BandgapBufferConfig ( PMC\_Type \* *base*, pmc\_bandgap\_buffer\_config\_t \* *config* ) [inline], [static]**

This function configures the PMC bandgap, including the drive select and behavior in low power mode.

### Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | Base address for current PMC instance. |
| <i>config</i> | Pointer to the configuration.          |

**49.2.4.8 static uint8\_t PMC\_HAL\_GetAckIsolation ( PMC\_Type \* *base* ) [inline], [static]**

This function reads the Acknowledge Isolation setting that indicates whether certain peripherals and the I/O pads are in a latched state as a result of having been in the VLLS mode.

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current PMC instance. |
|-------------|----------------------------------------|

### Returns

ACK isolation 0 - Peripherals and I/O pads are in a normal run state. 1 - Certain peripherals and I/O pads are in an isolated and latched state.

**49.2.4.9 static void PMC\_HAL\_ClearAckIsolation ( PMC\_Type \* *base* ) [inline], [static]**

This function clears the ACK Isolation flag. Writing one to this setting when it is set releases the I/O pads and certain peripherals to their normal run mode state.



Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current PMC instance. |
|-------------|----------------------------------------|

**49.2.4.10 static uint8\_t PMC\_HAL\_GetRegulatorStatus ( PMC\_Type \* *base* ) [inline], [static]**

This function returns the regulator to a run regulation status. It provides the current status of the internal voltage regulator.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current PMC instance. |
|-------------|----------------------------------------|

Returns

Regulation status 0 - Regulator is in a stop regulation or in transition to/from the regulation. 1 - Regulator is in a run regulation.



---

## Chapter 50

# Port Control and Interrupts (PORT)

### 50.1 Overview

The Kinetis SDK provides a HAL driver for the Port Control and Interrupts (PORT) block of Kinetis devices.

### Modules

- [PORT HAL driver](#)

## PORT HAL driver

### 50.2 PORT HAL driver

#### 50.2.1 Overview

The section describes the programming interface of the PORT 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, and so on.).

#### 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 = 0xCU }`  
*Digital filter clock source selection.*

## Configuration

- static void `PORT_HAL_SetMuxMode` (PORT\_Type \*base, uint32\_t pin, `port_mux_t` mux)  
*Configures the pin muxing.*
- void `PORT_HAL_SetLowGlobalPinCtrl` (PORT\_Type \*base, uint16\_t lowPinSelect, uint16\_t config)  
*Configures the low half of the pin control register for the same settings.*
- void `PORT_HAL_SetHighGlobalPinCtrl` (PORT\_Type \*base, uint16\_t highPinSelect, uint16\_t config)  
*Configures the high half of pin control register for the same settings.*

## Interrupt

- static void `PORT_HAL_SetPinIntMode` (PORT\_Type \*base, uint32\_t pin, `port_interrupt_config_t` intConfig)  
*Configures the port pin interrupt/DMA request.*
- static `port_interrupt_config_t` `PORT_HAL_GetPinIntMode` (PORT\_Type \*base, uint32\_t pin)  
*Gets the current port pin interrupt/DMA request configuration.*
- static bool `PORT_HAL_IsPinIntPending` (PORT\_Type \*base, uint32\_t pin)  
*Reads the individual pin-interrupt status flag.*
- static void `PORT_HAL_ClearPinIntFlag` (PORT\_Type \*base, uint32\_t pin)  
*Clears the individual pin-interrupt status flag.*
- static uint32\_t `PORT_HAL_GetPortIntFlag` (PORT\_Type \*base)  
*Reads the entire port interrupt status flag.*
- static void `PORT_HAL_ClearPortIntFlag` (PORT\_Type \*base)  
*Clears the entire port interrupt status flag.*

## 50.2.2 Enumeration Type Documentation

### 50.2.2.1 enum port\_pull\_t

Enumerator

***kPortPullDown*** internal pull-down resistor is enabled.  
***kPortPullUp*** internal pull-up resistor is enabled.

### 50.2.2.2 enum port\_slew\_rate\_t

Enumerator

***kPortFastSlewRate*** fast slew rate is configured.

## PORT HAL driver

*kPortSlowSlewRate* slow slew rate is configured.

### 50.2.2.3 enum port\_drive\_strength\_t

Enumerator

*kPortLowDriveStrength* low drive strength is configured.

*kPortHighDriveStrength* high drive strength is configured.

### 50.2.2.4 enum port\_mux\_t

Enumerator

*kPortPinDisabled* corresponding pin is disabled, but is used as an analog pin.

*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

### 50.2.2.5 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.

## 50.2.3 Function Documentation

**50.2.3.1** `static void PORT_HAL_SetMuxMode ( PORT_Type * base, uint32_t pin, port_mux_t mux ) [inline], [static]`

## Parameters

|             |                                                                                                                                                                                           |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | port base pointer                                                                                                                                                                         |
| <i>pin</i>  | port pin number                                                                                                                                                                           |
| <i>mux</i>  | pin muxing slot selection <ul style="list-style-type: none"> <li>• kPortPinDisabled: Pin disabled.</li> <li>• kPortMuxAsGpio : Set as GPIO.</li> <li>• others : chip-specific.</li> </ul> |

**50.2.3.2 void PORT\_HAL\_SetLowGlobalPinCtrl ( PORT\_Type \* *base*, uint16\_t *lowPinSelect*, uint16\_t *config* )**

This function operates pin 0 -15 of one specific port.

## Parameters

|                     |                                                                                                                                                                                                                                                                                                                                   |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | port base pointer                                                                                                                                                                                                                                                                                                                 |
| <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].                                                                                                                                                                                                                                                                  |

**50.2.3.3 void PORT\_HAL\_SetHighGlobalPinCtrl ( PORT\_Type \* *base*, uint16\_t *highPinSelect*, uint16\_t *config* )**

This function operates pin 16 -31 of one specific port.

## Parameters

|                      |                                                                                                                                                                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | port base pointer                                                                                                                                                                                                                                                                                                                   |
| <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].                                                                                                                                                                                                                                                                   |

## PORT HAL driver

**50.2.3.4 static void PORT\_HAL\_SetPinIntMode ( PORT\_Type \* *base*, uint32\_t *pin*, port\_interrupt\_config\_t *intConfig* ) [inline], [static]**

Parameters

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | port base pointer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <i>pin</i>       | port pin number                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <i>intConfig</i> | interrupt configuration <ul style="list-style-type: none"> <li>• kPortIntDisabled : Interrupt/DMA request disabled.</li> <li>• kPortDmaRisingEdge : DMA request on rising edge.</li> <li>• kPortDmaFallingEdge: DMA request on falling edge.</li> <li>• kPortDmaEitherEdge : DMA request on either edge.</li> <li>• kPortIntLogicZero : Interrupt when logic zero.</li> <li>• kPortIntRisingEdge : Interrupt on rising edge.</li> <li>• kPortIntFallingEdge: Interrupt on falling edge.</li> <li>• kPortIntEitherEdge : Interrupt on either edge.</li> <li>• kPortIntLogicOne : Interrupt when logic one.</li> </ul> |

**50.2.3.5 static port\_interrupt\_config\_t PORT\_HAL\_GetPinIntMode ( PORT\_Type \* *base*, uint32\_t *pin* ) [inline], [static]**

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | port base pointer |
| <i>pin</i>  | port pin number   |

Returns

interrupt configuration

- kPortIntDisabled : Interrupt/DMA request disabled.
- kPortDmaRisingEdge : DMA request on rising edge.
- kPortDmaFallingEdge: DMA request on falling edge.
- kPortDmaEitherEdge : DMA request on either edge.
- 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.



**50.2.3.6 static bool PORT\_HAL\_IsPinIntPending ( PORT\_Type \* *base*, 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>base</i> | port base pointer |
| <i>pin</i>  | port pin number   |

Returns

- current pin interrupt status flag
- 0: interrupt is not detected.
  - 1: interrupt is detected.

**50.2.3.7 static void PORT\_HAL\_ClearPinIntFlag ( PORT\_Type \* *base*, uint32\_t *pin* )  
[inline], [static]**

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | port base pointer |
| <i>pin</i>  | port pin number   |

**50.2.3.8 static uint32\_t PORT\_HAL\_GetPortIntFlag ( PORT\_Type \* *base* ) [inline],  
[static]**

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | port base pointer |
|-------------|-------------------|

Returns

- all 32 pin interrupt status flags. For specific bit:
- 0: interrupt is not detected.
  - 1: interrupt is detected.

**50.2.3.9 static void PORT\_HAL\_ClearPortIntFlag ( PORT\_Type \* *base* ) [inline],  
[static]**

---

## PORT HAL driver

Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | port base pointer |
|-------------|-------------------|

---

## Chapter 51

### Reset Control Module (RCM)

#### 51.1 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

### 51.2 RCM HAL driver

#### 51.2.1 Overview

The RCM implements the reset functions for the MCU.

RCM HAL driver provides these APIs:

- APIs for system reset source.
- APIs for RESET pin filter.
- APIs for ROM boot.

#### 51.2.2 System Reset Source APIs

The function [RCM\\_HAL\\_GetSrcStatus\(\)](#) checks whether the system is reset by specific source. For some platforms, there is sticky register for reset source. The function [RCM\\_HAL\\_GetStickySrcStatus\(\)](#) is used to check the sticky status and function [RCM\\_HAL\\_ClearStickySrcStatus\(\)](#) could clear the sticky status.

#### 51.2.3 RESET Pin Filter APIs

The function [RCM\\_HAL\\_SetResetPinFilterConfig\(\)](#) configures the RESET pin filter, including enable/disable filter, filter selection and filter width setting.

#### 51.2.4 ROM Boot APIs

The MCU could be configured to boot from ROM forcefully, use function [RCM\\_HAL\\_SetForceBootRomSrc\(\)](#) to set the boot source and use function [RCM\\_HAL\\_GetBootRomSrc\(\)](#) to get the boot source after reset.

### Files

- file [fsl\\_rcm\\_hal.h](#)

### Data Structures

- struct [rcm\\_reset\\_pin\\_filter\\_config\\_t](#)  
*Reset pin filter configuration. [More...](#)*

## Enumerations

- enum `rcm_source_names_t` {
  - `kRcmSrcAll` = 0U,
  - `kRcmWakeup` = RCM\_SRS0\_WAKEUP\_MASK,
  - `kRcmLowVoltDetect` = RCM\_SRS0\_LVD\_MASK,
  - `kRcmWatchDog` = RCM\_SRS0\_WDOG\_MASK,
  - `kRcmExternalPin` = RCM\_SRS0\_PIN\_MASK,
  - `kRcmPowerOn` = RCM\_SRS0\_POR\_MASK,
  - `kRcmCoreLockup` = RCM\_SRS1\_LOCKUP\_MASK << 8U,
  - `kRcmSoftware` = RCM\_SRS1\_SW\_MASK << 8U,
  - `kRcmMdmAp` = RCM\_SRS1\_MDM\_AP\_MASK << 8U,
  - `kRcmStopModeAckErr` = RCM\_SRS1\_SACKERR\_MASK << 8U }

*System Reset Source Name definitions.*
- enum `rcm_filter_run_wait_modes_t` {
  - `kRcmFilterDisabled`,
  - `kRcmFilterBusClk`,
  - `kRcmFilterLpoClk`,
  - `kRcmFilterReserverd` }

*Reset pin filter select in Run and Wait modes.*

## Reset Control Module APIs

- `uint32_t RCM_HAL_GetSrcStatus` (RCM\_Type \*base, `uint32_t` statusMask)
 

*Gets the reset source status.*
- `void RCM_HAL_SetResetPinFilterConfig` (RCM\_Type \*base, `rcm_reset_pin_filter_config_t` \*config)
 

*Sets the reset pin filter base on configuration.*

### 51.2.5 Data Structure Documentation

#### 51.2.5.1 struct `rcm_reset_pin_filter_config_t`

##### Data Fields

- `bool filterInStop`

*Reset pin filter select in stop mode.*
- `rcm_filter_run_wait_modes_t filterInRunWait`

*Reset pin filter in run/wait mode.*
- `uint8_t busClockFilterCount`

*Reset pin bus clock filter width.*

## RCM HAL driver

### 51.2.5.1.0.74 Field Documentation

51.2.5.1.0.74.1 `bool rcm_reset_pin_filter_config_t::filterInStop`

51.2.5.1.0.74.2 `rcm_filter_run_wait_modes_t rcm_reset_pin_filter_config_t::filterInRunWait`

51.2.5.1.0.74.3 `uint8_t rcm_reset_pin_filter_config_t::busClockFilterCount`

## 51.2.6 Enumeration Type Documentation

### 51.2.6.1 `enum rcm_source_names_t`

Enumerator

*kRcmSrcAll* Parameter could get all reset flags.

*kRcmWakeup* low-leakage wakeup reset

*kRcmLowVoltDetect* low voltage detect reset

*kRcmWatchDog* watch dog reset

*kRcmExternalPin* external pin reset

*kRcmPowerOn* power on reset

*kRcmCoreLockup* core lockup reset

*kRcmSoftware* software reset

*kRcmMdmAp* MDM-AP system reset.

*kRcmStopModeAckErr* stop mode ACK error reset

### 51.2.6.2 `enum rcm_filter_run_wait_modes_t`

Enumerator

*kRcmFilterDisabled* all filtering disabled

*kRcmFilterBusClk* Bus clock filter enabled.

*kRcmFilterLpoClk* LPO clock filter enabled.

*kRcmFilterReserverd* reserved setting

## 51.2.7 Function Documentation

### 51.2.7.1 `uint32_t RCM_HAL_GetSrcStatus ( RCM_Type * base, uint32_t statusMask )`

This function gets the current reset source status for some specified sources.

Example:

```
uint32_t resetStatus;

// To get all reset source statuses.
resetStatus = RCM_HAL_GetSrcStatus(RCM, kRcmSrcAll);
```

```
// To test whether MCU is reset by watchdog.
resetStatus = RCM_HAL_GetSrcStatus(RCM, kRcmWatchDog);

// To test multiple reset source.
resetStatus = RCM_HAL_GetSrcStatus(RCM, kRcmWatchDog |
 kRcmSoftware);
```

Parameters

|                   |                                        |
|-------------------|----------------------------------------|
| <i>base</i>       | Register base address of RCM           |
| <i>statusMask</i> | Bit mask for the reset sources to get. |

Returns

The reset source status.

**51.2.7.2 void RCM\_HAL\_SetResetPinFilterConfig ( RCM\_Type \* *base*,  
rcm\_reset\_pin\_filter\_config\_t \* *config* )**

This function sets the reset pin filter, including filter source, filter width and so on.

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | Register base address of RCM            |
| <i>config</i> | Pointer to the configuration structure. |





---

## Chapter 52 System Clock Generator (SCG)

### 52.1 Overview

The Kinetis SDK provides a HAL driver for the System Clock Generator (SCG) block of Kinetis devices.

### Modules

- [SCG HAL driver](#)

## SCG HAL driver

### 52.2 SCG HAL driver

This chapter describes the programming interface of the SCG HAL driver. The SCG HAL driver configures the SCG (System Clock Generator). It handles calibration, initialization, and configuration of SCG module.

SCG driver provides three kinds of APIs:

1. APIs for MCU system clock.
2. APIs for clock out selection.
3. APIs for clock source configuration, including system OSC, slow IRC, fast IRC and system PLL.

#### 52.2.1 APIs for MCU system clock

MCU system clock configuration includes clock source and dividers. There are dedicated control registers for RUN mode, VLPR mode and HSRUN mode. When MCU switches to new power mode, the configuration for corresponding mode will be used.

SCG driver provides the APIs to set and get this configure registers, for example, to get the system clock configuration of RUN mode, please use the function `CLOCK_HAL_GetSystemClockConfigInRun()`, to set the system clock configuration of VLPR mode, please use the function `CLOCK_HAL_SetSystemClockConfigInVlpr()`.

Besides these APIs, SCG HAL driver also provides `CLOCK_HAL_GetSystemClockFreq()` to get the system clock output frequency.

#### 52.2.2 APIs for clock out selection

For SCG CLKOUT, SCG HAL driver provides `CLOCK_HAL_GetClkOutSel()` and `CLOCK_HAL_SetClkOutSel()` to get and set configuration.

#### 52.2.3 APIs for clock source configuration

SCG has four clock sources, system OSC, system PLL, fast IRC and slow IRC. For each clock source, SCG driver provides such APIs:

- API to setup the clock source one step base on configuration structure.
- API to get the default configuration structure. This configuration structure could make the clock work, for special use case, please modify this structure.
- API to validate the configuration structure (optional).
- API to get the frequency of this clock source.
- API to get the asynchronous frequency of this clock source.
- API to de-initialize the clock source.

There is an example shows how to setup the clock source base on SCG HAL APIs.

```
/* Example code for how to setup system OSC. */

scg_sys_osc_config_t config;

// Get default configuration.
CLOCK_HAL_GetSysOscDefaultConfig(&config);

// Modify according to board setting.
// ...

// Setup system OSC.
CLOCK_HAL_InitSysOsc(SCG_BASE, &config);

// De-init system OSC.
CLOCK_HAL_DeinitSysOsc(SCG_BASE);
```

Because some configurations of the clock source could only be changed when the clock source is disabled, so the function `CLOCK_HAL_InitXxx` will disable the clock source, then re-configure it and enable it. As a result, before `CLOCK_HAL_InitXxx`, please make sure the clock source is not used.

- Not chosen as the system clock.
- Not used to generate asynchronous clock for on-chip peripherals.
- Not used by other clock source. For example, FIRC could be used as the clock source of system PLL, so when system PLL is used, it is not allowed to use `CLOCK_HAL_InitFirc` to re-configure FIRC.



---

## Chapter 53 System Integration Module (SIM)

### 53.1 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

### 53.2 SIM HAL driver

#### 53.2.1 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.

#### 53.2.2 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);
```

#### 53.2.3 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) //
 kClockUsbfsSrcPllFlSel, // 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.

### 53.2.4 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 usbdiv,
 uint8_t usbfrac);

// To get USB FS OTG divider setting. //
void CLOCK_HAL_GetUsbfsDiv(uint32_t baseAddr,
 uint8_t *usbdiv,
 uint8_t *usbfrac);
```

### Modules

- [K02F12810 SIM HAL driver](#)
- [K10D10 SIM HAL driver](#)
- [K11DA5 SIM HAL driver](#)
- [K20D10 SIM HAL driver](#)
- [K21DA5 SIM HAL driver](#)
- [K21FA12 SIM HAL driver](#)
- [K22F12810 SIM HAL driver](#)
- [K22F25612 SIM HAL driver](#)
- [K22F51212 SIM HAL driver](#)
- [K24F12 SIM HAL driver](#)
- [K24F25612 SIM HAL driver](#)
- [K26F18 SIM HAL driver](#)
- [K30D10 SIM HAL driver](#)
- [K40D10 SIM HAL driver](#)
- [K50D10 SIM HAL driver](#)
- [K51D10 SIM HAL driver](#)
- [K52D10 SIM HAL driver](#)
- [K53D10 SIM HAL driver](#)
- [K60D10 SIM HAL driver](#)
- [K63F12 SIM HAL driver](#)
- [K64F12 SIM HAL driver](#)
- [K65F18 SIM HAL driver](#)
- [K66F18 SIM HAL driver](#)

## SIM HAL driver

- [KL02Z4 SIM HAL driver](#)
- [KL03Z4 SIM HAL driver](#)
- [KL13Z644 SIM HAL driver](#)
- [KL16Z4 SIM HAL driver](#)
- [KL17Z4 SIM HAL driver](#)
- [KL17Z644 SIM HAL driver](#)
- [KL26Z4 SIM HAL driver](#)
- [KL27Z4 SIM HAL driver](#)
- [KL27Z644 SIM HAL driver](#)
- [KL28T7 SIM HAL driver](#)
- [KL33Z4 SIM HAL driver](#)
- [KL33Z644 SIM HAL driver](#)
- [KL34Z4 SIM HAL driver](#)
- [KL36Z4 SIM HAL driver](#)
- [KL43Z4 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](#)
- [KW01Z4 SIM HAL driver](#)
- [KW21D5 SIM HAL driver](#)
- [KW22D5 SIM HAL driver](#)
- [KW24D5 SIM HAL driver](#)

## Files

- [file fsl\\_sim\\_hal.h](#)
- [file fsl\\_sim\\_hal\\_MKL14Z4.h](#)
- [file fsl\\_sim\\_hal\\_MKL15Z4.h](#)
- [file fsl\\_sim\\_hal\\_MKL24Z4.h](#)
- [file fsl\\_sim\\_hal\\_MKL25Z4.h](#)
- [file fsl\\_sim\\_hal\\_MKV40F15.h](#)
- [file fsl\\_sim\\_hal\\_MKV43F15.h](#)
- [file fsl\\_sim\\_hal\\_MKV44F15.h](#)
- [file fsl\\_sim\\_hal\\_MKV45F15.h](#)
- [file fsl\\_sim\\_hal\\_MKV46F15.h](#)

## Macros

- `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`  
*SIM SCGC bit index.*
- `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`  
*SIM SCGC bit index.*
- `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`  
*SIM SCGC bit index.*
- `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`  
*SIM SCGC bit index.*
- `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`  
*SIM SCGC bit index.*



## Enumerations

- enum `sim_hal_status_t` {  
`kSimHalSuccess`,  
`kSimHalFail` }  
*SIM HAL API return status.*
- enum `clock_cop_src_t`  
*COP clock source select.*
- enum `clock_tpm_src_t`  
*TPM clock source select.*
- enum `clock_lptmr_src_t` { ,  
`kClockLptmrSrcMcgIrClk`,  
`kClockLptmrSrcLpoClk`,  
`kClockLptmrSrcEr32kClk`,  
`kClockLptmrSrcOsc0erClkUndiv`,  
`kClockLptmrSrcMcgIrClk`,  
`kClockLptmrSrcLpoClk`,  
`kClockLptmrSrcEr32kClk`,  
`kClockLptmrSrcOsc0erClkUndiv`,  
`kClockLptmrSrcMcgIrClk`,  
`kClockLptmrSrcLpoClk`,  
`kClockLptmrSrcEr32kClk`,  
`kClockLptmrSrcOsc0erClkUndiv`,  
`kClockLptmrSrcMcgIrClk`,  
`kClockLptmrSrcLpoClk`,  
`kClockLptmrSrcEr32kClk`,  
`kClockLptmrSrcOsc0erClkUndiv`,  
`kClockLptmrSrcMcgIrClk`,  
`kClockLptmrSrcLpoClk`,  
`kClockLptmrSrcEr32kClk`,  
`kClockLptmrSrcOsc0erClkUndiv` }  
*LPTMR clock source select.*
- enum `clock_lpsci_src_t`  
*UART0 clock source select.*
- enum `clock_pllfl1_sel_t`  
*USB clock source select.*
- enum `clock_er32k_src_t` { ,  
`kClockEr32kSrcOsc0 = 0U`,  
`kClockEr32kSrcLpo = 3U`,  
`kClockEr32kSrcOsc0 = 0U`,  
`kClockEr32kSrcLpo = 3U`,  
`kClockEr32kSrcOsc0 = 0U`,  
`kClockEr32kSrcLpo = 3U`,  
`kClockEr32kSrcOsc0 = 0U`,  
`kClockEr32kSrcLpo = 3U`,  
`kClockEr32kSrcOsc0 = 0U`,

## SIM HAL driver

- ```

kClockEr32kSrcLpo = 3U }
    SIM external reference clock source select (OSC32KSEL)
• enum clock_clkout_src_t
    SIM CLKOUT_SEL clock source select.
• enum clock_rtcout_src_t
    SIM RTCCLKOUTSEL clock source select.
• enum sim_adc_pretrg_sel_t
    SIM USB voltage regulator in standby mode setting during stop modes.
• enum sim_adc_trg_sel_t
    SIM ADCx trigger select.
• enum sim_uart_rxsrc_t
    SIM UART receive data source select.
• enum sim_uart_txsrc_t
    SIM UART transmit data source select.
• enum sim_lpsci_rxsrc_t
    SIM LPSCI receive data source select.
• enum sim_lpsci_txsrc_t
    SIM LPSCI transmit data source 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.
• enum clock_cop_src_t
    COP clock source select.
• enum clock_tpm_src_t
    TPM clock source select.
• enum clock_lptmr_src_t { ,
kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv,
kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv,
kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv,
kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv,
kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv }
    LPTMR clock source select.

```

- enum `clock_lpisci_src_t`
UART0 clock source select.
- enum `clock_pllflr_sel_t`
USB clock source select.
- enum `clock_er32k_src_t` { ,
`kClockEr32kSrcOsc0` = 0U,
`kClockEr32kSrcLpo` = 3U,
`kClockEr32kSrcOsc0` = 0U,
`kClockEr32kSrcLpo` = 3U,
`kClockEr32kSrcOsc0` = 0U,
`kClockEr32kSrcLpo` = 3U,
`kClockEr32kSrcOsc0` = 0U,
`kClockEr32kSrcLpo` = 3U,
`kClockEr32kSrcOsc0` = 0U,
`kClockEr32kSrcLpo` = 3U }
- *SIM external reference clock source select (OSC32KSEL)*
- enum `clock_clkout_src_t`
SIM CLKOUT_SEL clock source select.
- enum `clock_rtcout_src_t`
SIM RTCCLKOUTSEL clock source select.
- enum `sim_adc_pretrg_sel_t`
SIM USB voltage regulator in standby mode setting during stop modes.
- enum `sim_adc_trg_sel_t`
SIM ADCx trigger select.
- enum `sim_uart_rxsrc_t`
SIM UART receive data source select.
- enum `sim_uart_txsrc_t`
SIM UART transmit data source select.
- enum `sim_lpisci_rxsrc_t`
SIM LPSCI receive data source select.
- enum `sim_lpisci_txsrc_t`
SIM LPSCI transmit data source 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.
- enum `clock_cop_src_t`
COP clock source select.
- enum `clock_tpm_src_t`
TPM clock source select.
- enum `clock_lptmr_src_t` { ,

SIM HAL driver

```

kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv,
kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv,
kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv,
kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv,
kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv }

```

LPTMR clock source select.

- enum `clock_lpsci_src_t`

UART0 clock source select.

- enum `clock_pllfl1_sel_t`

USB clock source select.

- enum `clock_er32k_src_t` { ,
`kClockEr32kSrcOsc0 = 0U,`
`kClockEr32kSrcLpo = 3U,`
`kClockEr32kSrcOsc0 = 0U,`
`kClockEr32kSrcLpo = 3U,`
`kClockEr32kSrcOsc0 = 0U,`
`kClockEr32kSrcLpo = 3U,`
`kClockEr32kSrcOsc0 = 0U,`
`kClockEr32kSrcLpo = 3U,`
`kClockEr32kSrcOsc0 = 0U,`
`kClockEr32kSrcLpo = 3U` }

SIM external reference clock source select (OSC32KSEL)

- enum `clock_clkout_src_t`

SIM CLKOUT_SEL clock source select.

- enum `clock_rtcout_src_t`

SIM RTCCLKOUTSEL clock source select.

- enum `sim_adc_pretrg_sel_t`

SIM USB voltage regulator in standby mode setting during stop modes.

- enum `sim_adc_trg_sel_t`

SIM ADCx trigger select.

- enum `sim_uart_rxsrc_t`

SIM UART receive data source select.

- enum [sim_uart_txsrc_t](#)
SIM UART transmit data source select.
- enum [sim_lpsci_rxsrc_t](#)
SIM LPSCI receive data source select.
- enum [sim_lpsci_txsrc_t](#)
SIM LPSCI transmit data source 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.
- enum [clock_cop_src_t](#)
COP clock source select.
- enum [clock_tpm_src_t](#)
TPM clock source select.
- enum [clock_lptmr_src_t](#) { ,
[kClockLptmrSrcMcgIrClk](#),
[kClockLptmrSrcLpoClk](#),
[kClockLptmrSrcEr32kClk](#),
[kClockLptmrSrcOsc0erClkUndiv](#),
[kClockLptmrSrcMcgIrClk](#),
[kClockLptmrSrcLpoClk](#),
[kClockLptmrSrcEr32kClk](#),
[kClockLptmrSrcOsc0erClkUndiv](#),
[kClockLptmrSrcMcgIrClk](#),
[kClockLptmrSrcLpoClk](#),
[kClockLptmrSrcEr32kClk](#),
[kClockLptmrSrcOsc0erClkUndiv](#),
[kClockLptmrSrcMcgIrClk](#),
[kClockLptmrSrcLpoClk](#),
[kClockLptmrSrcEr32kClk](#),
[kClockLptmrSrcOsc0erClkUndiv](#),
[kClockLptmrSrcMcgIrClk](#),
[kClockLptmrSrcLpoClk](#),
[kClockLptmrSrcEr32kClk](#),
[kClockLptmrSrcOsc0erClkUndiv](#) }
- *LPTMR clock source select.*
- enum [clock_lpsci_src_t](#)
UART0 clock source select.
- enum [clock_pllfl_sel_t](#)
USB clock source select.
- enum [clock_er32k_src_t](#) { ,

SIM HAL driver

- ```

kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U }

```
- *SIM external reference clock source select (OSC32KSEL)*
  - enum `clock_clkout_src_t`  
*SIM CLKOUT\_SEL clock source select.*
  - enum `clock_rtcout_src_t`  
*SIM RTCCLKOUTSEL clock source select.*
  - enum `sim_adc_pretrg_sel_t`  
*SIM USB voltage regulator in standby mode setting during stop modes.*
  - enum `sim_adc_trg_sel_t`  
*SIM ADCx trigger select.*
  - enum `sim_uart_rxsrc_t`  
*SIM UART receive data source select.*
  - enum `sim_uart_txsrc_t`  
*SIM UART transmit data source select.*
  - enum `sim_lpsci_rxsrc_t`  
*SIM LPSCI receive data source select.*
  - enum `sim_lpsci_txsrc_t`  
*SIM LPSCI transmit data source 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.*
  - enum `clock_wdog_src_t` {  
`kClockWdogSrcLpoClk,`  
`kClockWdogSrcAltClk,`  
`kClockWdogSrcLpoClk,`  
`kClockWdogSrcAltClk,`  
`kClockWdogSrcLpoClk,`  
`kClockWdogSrcAltClk,`  
`kClockWdogSrcLpoClk,`  
`kClockWdogSrcAltClk,`  
`kClockWdogSrcLpoClk,`  
`kClockWdogSrcAltClk` }  
*WDOG clock source select.*
  - enum `clock_trace_src_t` {

```

kClockTraceSrcMcgoutClk,
kClockTraceSrcCoreClk,
kClockTraceSrcMcgoutClk,
kClockTraceSrcCoreClk,
kClockTraceSrcMcgoutClk,
kClockTraceSrcCoreClk,
kClockTraceSrcMcgoutClk,
kClockTraceSrcCoreClk,
kClockTraceSrcMcgoutClk,
kClockTraceSrcCoreClk }

```

*Debug trace clock source select.*

- enum `clock_nanoedge_clk2x_src` {

```

kClockNanoedgeSrcMcgPllClk,
kClockNanoedgeSrcMcgPllClk2x,
kClockNanoedgeSrcMcgPllClk,
kClockNanoedgeSrcMcgPllClk2x,
kClockNanoedgeSrcMcgPllClk,
kClockNanoedgeSrcMcgPllClk2x,
kClockNanoedgeSrcMcgPllClk,
kClockNanoedgeSrcMcgPllClk2x,
kClockNanoedgeSrcMcgPllClk,
kClockNanoedgeSrcMcgPllClk2x }

```

*Debug trace clock source select.*

- enum `sim_osc32k_clock_sel_t`

*SIM OSC32KSEL clock source select.*
- enum `sim_nanoedge_clock_sel_t`

*SIM NANOEDGECLK2XSEL clock source select.*
- enum `sim_trace_clock_sel_t`

*SIM TRACECLKSEL clock source select.*
- enum `sim_clkout_clock_sel_t`

*SIM CLKOUT\_SEL clock source select.*
- enum `sim_adcb_trg_sel_t`

*SIM ADCB trigger select.*
- enum `sim_adc_trg_sel_t`

*SIM ADC trigger select.*
- enum `sim_cadc_conv_id_t` {

```

kSimCAdcConvA = 0U,
kSimCAdcConvB = 1U,
kSimCAdcConvA = 0U,
kSimCAdcConvB = 1U,
kSimCAdcConvA = 0U,
kSimCAdcConvB = 1U,
kSimCAdcConvA = 0U,
kSimCAdcConvB = 1U,
kSimCAdcConvA = 0U,
kSimCAdcConvB = 1U,
kSimCAdcConvA = 0U,
kSimCAdcConvB = 1U }

```

*Defines the type of enumerating ADC converter's ID.*

## SIM HAL driver

- enum `sim_adc_alt_trg_en`  
*SIM ADC alternate trigger enable.*
- enum `sim_dac_hw_trg_sel`  
*DAC0 Hardware Trigger Input Source.*
- enum `sim_ewm_in_src`  
*the ewm\_in source of EWM module.*
- enum `sim_cmp_win_in_src`  
*CMP Sample/Window Input X Source.*
- enum `clock_lptmr_src_t` { ,  
`kClockLptmrSrcMcgIrClk,`  
`kClockLptmrSrcLpoClk,`  
`kClockLptmrSrcEr32kClk,`  
`kClockLptmrSrcOsc0erClkUndiv,`  
`kClockLptmrSrcMcgIrClk,`  
`kClockLptmrSrcLpoClk,`  
`kClockLptmrSrcEr32kClk,`  
`kClockLptmrSrcOsc0erClkUndiv,`  
`kClockLptmrSrcMcgIrClk,`  
`kClockLptmrSrcLpoClk,`  
`kClockLptmrSrcEr32kClk,`  
`kClockLptmrSrcOsc0erClkUndiv,`  
`kClockLptmrSrcMcgIrClk,`  
`kClockLptmrSrcLpoClk,`  
`kClockLptmrSrcEr32kClk,`  
`kClockLptmrSrcOsc0erClkUndiv,`  
`kClockLptmrSrcMcgIrClk,`  
`kClockLptmrSrcLpoClk,`  
`kClockLptmrSrcEr32kClk,`  
`kClockLptmrSrcOsc0erClkUndiv` }  
*LPTMR clock source select.*
- enum `clock_er32k_src_t` { ,  
`kClockEr32kSrcOsc0 = 0U,`  
`kClockEr32kSrcLpo = 3U,`  
`kClockEr32kSrcOsc0 = 0U,`  
`kClockEr32kSrcLpo = 3U,`  
`kClockEr32kSrcOsc0 = 0U,`  
`kClockEr32kSrcLpo = 3U,`  
`kClockEr32kSrcOsc0 = 0U,`  
`kClockEr32kSrcLpo = 3U,`  
`kClockEr32kSrcOsc0 = 0U,`  
`kClockEr32kSrcLpo = 3U` }  
*SIM external reference clock source select (OSC32KSEL).*
- enum `clock_flexcan_src_t` {



```

kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk,
kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk,
kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk,
kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk,
kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk }

```

*FLEXCAN clock source select.*

- enum `sim_clock_gate_name_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*
- enum `clock_source_names_t`  
*Clock source and sel names.*
- enum `clock_divider_names_t`  
*Clock Divider names.*
- enum `sim_usbstby_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_flexbus_security_level_t`  
*SIM FlexBus security level.*
- enum `sim_uart_rxsrc_t`  
*SIM UART receive data source select.*
- enum `sim_uart_txsrc_t`  
*SIM UART 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_ch_out_src_t`  
*SIM FlexTimer x channel y output source select.*
- enum `sim_ftm_ft_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.*
- enum `clock_wdog_src_t` {

## SIM HAL driver

- kClockWdogSrcLpoClk,
- kClockWdogSrcAltClk,
- kClockWdogSrcLpoClk,
- kClockWdogSrcAltClk,
- kClockWdogSrcLpoClk,
- kClockWdogSrcAltClk,
- kClockWdogSrcLpoClk,
- kClockWdogSrcAltClk,
- kClockWdogSrcLpoClk,
- kClockWdogSrcAltClk }
- WDOG clock source select.*
- enum `clock_trace_src_t` {
  - kClockTraceSrcMcgoutClk,
  - kClockTraceSrcCoreClk,
  - kClockTraceSrcMcgoutClk,
  - kClockTraceSrcCoreClk,
  - kClockTraceSrcMcgoutClk,
  - kClockTraceSrcCoreClk,
  - kClockTraceSrcMcgoutClk,
  - kClockTraceSrcCoreClk,
  - kClockTraceSrcMcgoutClk,
  - kClockTraceSrcCoreClk }
  - Debug trace clock source select.*
  - enum `clock_nanoedge_clk2x_src` {
    - kClockNanoedgeSrcMcgPllClk,
    - kClockNanoedgeSrcMcgPllClk2x,
    - kClockNanoedgeSrcMcgPllClk,
    - kClockNanoedgeSrcMcgPllClk2x,
    - kClockNanoedgeSrcMcgPllClk,
    - kClockNanoedgeSrcMcgPllClk2x,
    - kClockNanoedgeSrcMcgPllClk,
    - kClockNanoedgeSrcMcgPllClk2x,
    - kClockNanoedgeSrcMcgPllClk,
    - kClockNanoedgeSrcMcgPllClk2x }
    - Debug trace clock source select.*
    - enum `sim_osc32k_clock_sel_t`
      - SIM OSC32KSEL clock source select.*
    - enum `sim_nanoedge_clock_sel_t`
      - SIM NANOEDGECLK2XSEL clock source select.*
    - enum `sim_trace_clock_sel_t`
      - SIM TRACECLKSEL clock source select.*
    - enum `sim_clkout_clock_sel_t`
      - SIM CLKOUT\_SEL clock source select.*
    - enum `sim_adcb_trg_sel_t`
      - SIM ADCB trigger select.*
    - enum `sim_adc_trg_sel_t`
      - SIM ADC trigger select.*

- enum `sim_cadc_conv_id_t` {
  - `kSimCAdcConvA = 0U,`
  - `kSimCAdcConvB = 1U,`
  - `kSimCAdcConvA = 0U,`
  - `kSimCAdcConvB = 1U,`
  - `kSimCAdcConvA = 0U,`
  - `kSimCAdcConvB = 1U,`
  - `kSimCAdcConvA = 0U,`
  - `kSimCAdcConvB = 1U,`
  - `kSimCAdcConvA = 0U,`
  - `kSimCAdcConvB = 1U }`

*Defines the type of enumerating ADC converter's ID.*
- enum `sim_adc_alt_trg_en`

*SIM ADC alternate trigger enable.*
- enum `sim_dac_hw_trg_sel`

*DAC0 Hardware Trigger Input Source.*
- enum `sim_ewm_in_src`

*the ewm\_in source of EWM module.*
- enum `sim_cmp_win_in_src`

*CMP Sample/Window Input X Source.*
- enum `clock_lptmr_src_t` { ,
  - `kClockLptmrSrcMcgIrClk,`
  - `kClockLptmrSrcLpoClk,`
  - `kClockLptmrSrcEr32kClk,`
  - `kClockLptmrSrcOsc0erClkUndiv,`
  - `kClockLptmrSrcMcgIrClk,`
  - `kClockLptmrSrcLpoClk,`
  - `kClockLptmrSrcEr32kClk,`
  - `kClockLptmrSrcOsc0erClkUndiv,`
  - `kClockLptmrSrcMcgIrClk,`
  - `kClockLptmrSrcLpoClk,`
  - `kClockLptmrSrcEr32kClk,`
  - `kClockLptmrSrcOsc0erClkUndiv,`
  - `kClockLptmrSrcMcgIrClk,`
  - `kClockLptmrSrcLpoClk,`
  - `kClockLptmrSrcEr32kClk,`
  - `kClockLptmrSrcOsc0erClkUndiv,`
  - `kClockLptmrSrcMcgIrClk,`
  - `kClockLptmrSrcLpoClk,`
  - `kClockLptmrSrcEr32kClk,`
  - `kClockLptmrSrcOsc0erClkUndiv }`

*LPTMR clock source select.*
- enum `clock_er32k_src_t` { ,

## SIM HAL driver

```

kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U }

```

*SIM external reference clock source select (OSC32KSEL).*

- enum `clock_flexcan_src_t` {
  - `kClockFlexcanSrcOsc0erClk,`
  - `kClockFlexcanSrcBusClk,`
  - `kClockFlexcanSrcOsc0erClk,`
  - `kClockFlexcanSrcBusClk,`
  - `kClockFlexcanSrcOsc0erClk,`
  - `kClockFlexcanSrcBusClk,`
  - `kClockFlexcanSrcOsc0erClk,`
  - `kClockFlexcanSrcBusClk,`
  - `kClockFlexcanSrcOsc0erClk,`
  - `kClockFlexcanSrcBusClk` }

*FLEXCAN clock source select.*
- enum `sim_clock_gate_name_t`

*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*
- enum `clock_source_names_t`

*Clock source and sel names.*
- enum `clock_divider_names_t`

*Clock Divider names.*
- enum `sim_usbstby_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_flexbus_security_level_t`

*SIM FlexBus security level.*
- enum `sim_uart_rxsrt_t`

*SIM UART receive data source select.*
- enum `sim_uart_txsrc_t`

*SIM UART 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_ch_out_src_t`
- SIM FlexTimer x channel y output source select.*

  - enum `sim_ftm_ft_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.*

  - enum `clock_wdog_src_t` {  
`kClockWdogSrcLpoClk,`  
`kClockWdogSrcAltClk,`  
`kClockWdogSrcLpoClk,`  
`kClockWdogSrcAltClk,`  
`kClockWdogSrcLpoClk,`  
`kClockWdogSrcAltClk,`  
`kClockWdogSrcLpoClk,`  
`kClockWdogSrcAltClk,`  
`kClockWdogSrcLpoClk,`  
`kClockWdogSrcAltClk,`  
`kClockWdogSrcLpoClk,`  
`kClockWdogSrcAltClk` }

*WDOG clock source select.*

  - enum `clock_trace_src_t` {  
`kClockTraceSrcMcgoutClk,`  
`kClockTraceSrcCoreClk,`  
`kClockTraceSrcMcgoutClk,`  
`kClockTraceSrcCoreClk,`  
`kClockTraceSrcMcgoutClk,`  
`kClockTraceSrcCoreClk,`  
`kClockTraceSrcMcgoutClk,`  
`kClockTraceSrcCoreClk,`  
`kClockTraceSrcMcgoutClk,`  
`kClockTraceSrcCoreClk,`  
`kClockTraceSrcMcgoutClk,`  
`kClockTraceSrcCoreClk` }

*Debug trace clock source select.*

  - enum `clock_nanoedge_clk2x_src` {  
`kClockNanoedgeSrcMcgPllClk,`  
`kClockNanoedgeSrcMcgPllClk2x,`  
`kClockNanoedgeSrcMcgPllClk,`  
`kClockNanoedgeSrcMcgPllClk2x,`  
`kClockNanoedgeSrcMcgPllClk,`  
`kClockNanoedgeSrcMcgPllClk2x,`  
`kClockNanoedgeSrcMcgPllClk,`  
`kClockNanoedgeSrcMcgPllClk2x,`  
`kClockNanoedgeSrcMcgPllClk,`  
`kClockNanoedgeSrcMcgPllClk2x,`  
`kClockNanoedgeSrcMcgPllClk,`  
`kClockNanoedgeSrcMcgPllClk2x` }

*Debug trace clock source select.*

  - enum `sim_osc32k_clock_sel_t`

*SIM OSC32KSEL clock source select.*

## SIM HAL driver

- enum [sim\\_nanoedge\\_clock\\_sel\\_t](#)  
*SIM NANOEDGECLK2XSEL clock source select.*
- enum [sim\\_trace\\_clock\\_sel\\_t](#)  
*SIM TRACECLKSEL clock source select.*
- enum [sim\\_clkout\\_clock\\_sel\\_t](#)  
*SIM CLKOUT\_SEL clock source select.*
- enum [sim\\_adcb\\_trg\\_sel\\_t](#)  
*SIM ADCB trigger select.*
- enum [sim\\_adc\\_trg\\_sel\\_t](#)  
*SIM ADC trigger select.*
- enum [sim\\_cadc\\_conv\\_id\\_t](#) {  
[kSimCAdcConvA = 0U,](#)  
[kSimCAdcConvB = 1U,](#)  
[kSimCAdcConvA = 0U,](#)  
[kSimCAdcConvB = 1U,](#)  
[kSimCAdcConvA = 0U,](#)  
[kSimCAdcConvB = 1U,](#)  
[kSimCAdcConvA = 0U,](#)  
[kSimCAdcConvB = 1U,](#)  
[kSimCAdcConvA = 0U,](#)  
[kSimCAdcConvB = 1U }  
\*Defines the type of enumerating ADC converter's ID.\*](#)
- enum [sim\\_adc\\_alt\\_trg\\_en](#)  
*SIM ADC alternate trigger enable.*
- enum [sim\\_dac\\_hw\\_trg\\_sel](#)  
*DAC0 Hardware Trigger Input Source.*
- enum [sim\\_ewm\\_in\\_src](#)  
*the ewm\_in source of EWM module.*
- enum [sim\\_cmp\\_win\\_in\\_src](#)  
*CMP Sample/Window Input X Source.*
- enum [clock\\_lptmr\\_src\\_t](#) { ,

```

kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv,
kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv,
kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv,
kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv,
kClockLptmrSrcMcgIrClk,
kClockLptmrSrcLpoClk,
kClockLptmrSrcEr32kClk,
kClockLptmrSrcOsc0erClkUndiv }

```

*LPTMR clock source select.*

- enum clock\_er32k\_src\_t { ,
 

```

kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U }

```

*SIM external reference clock source select (OSC32KSEL).*

- enum clock\_flexcan\_src\_t {
 

```

kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk,
kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk,
kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk,
kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk,
kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk }

```

*FLEXCAN clock source select.*

## SIM HAL driver

- enum [sim\\_clock\\_gate\\_name\\_t](#)  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*
- enum [clock\\_source\\_names\\_t](#)  
*Clock source and sel names.*
- enum [clock\\_divider\\_names\\_t](#)  
*Clock Divider names.*
- 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\\_flexbus\\_security\\_level\\_t](#)  
*SIM FlexBus security level.*
- enum [sim\\_uart\\_rxsrc\\_t](#)  
*SIM UART receive data source select.*
- enum [sim\\_uart\\_txsrc\\_t](#)  
*SIM UART 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\\_ch\\_out\\_src\\_t](#)  
*SIM FlexTimer x channel y output source select.*
- enum [sim\\_ftmflt\\_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.*
- enum [clock\\_wdog\\_src\\_t](#) {  
  [kClockWdogSrcLpoClk](#),  
  [kClockWdogSrcAltClk](#),  
  [kClockWdogSrcLpoClk](#),  
  [kClockWdogSrcAltClk](#),  
  [kClockWdogSrcLpoClk](#),  
  [kClockWdogSrcAltClk](#),  
  [kClockWdogSrcLpoClk](#),  
  [kClockWdogSrcAltClk](#),  
  [kClockWdogSrcLpoClk](#),  
  [kClockWdogSrcAltClk](#) }  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_t](#) {



```

kClockTraceSrcMcgoutClk,
kClockTraceSrcCoreClk,
kClockTraceSrcMcgoutClk,
kClockTraceSrcCoreClk,
kClockTraceSrcMcgoutClk,
kClockTraceSrcCoreClk,
kClockTraceSrcMcgoutClk,
kClockTraceSrcCoreClk,
kClockTraceSrcMcgoutClk,
kClockTraceSrcCoreClk }

```

*Debug trace clock source select.*

- enum `clock_nanoedge_clk2x_src` {
 

```

kClockNanoedgeSrcMcgPllClk,
kClockNanoedgeSrcMcgPllClk2x,
kClockNanoedgeSrcMcgPllClk,
kClockNanoedgeSrcMcgPllClk2x,
kClockNanoedgeSrcMcgPllClk,
kClockNanoedgeSrcMcgPllClk2x,
kClockNanoedgeSrcMcgPllClk,
kClockNanoedgeSrcMcgPllClk2x,
kClockNanoedgeSrcMcgPllClk,
kClockNanoedgeSrcMcgPllClk2x }

```

*Debug trace clock source select.*

- enum `sim_osc32k_clock_sel_t`

*SIM OSC32KSEL clock source select.*
- enum `sim_nanoedge_clock_sel_t`

*SIM NANOEDGECLK2XSEL clock source select.*
- enum `sim_trace_clock_sel_t`

*SIM TRACECLKSEL clock source select.*
- enum `sim_clkout_clock_sel_t`

*SIM CLKOUT\_SEL clock source select.*
- enum `sim_adcb_trg_sel_t`

*SIM ADCB trigger select.*
- enum `sim_adc_trg_sel_t`

*SIM ADC trigger select.*
- enum `sim_cadc_conv_id_t` {
 

```

kSimCAdcConvA = 0U,
kSimCAdcConvB = 1U,
kSimCAdcConvA = 0U,
kSimCAdcConvB = 1U,
kSimCAdcConvA = 0U,
kSimCAdcConvB = 1U,
kSimCAdcConvA = 0U,
kSimCAdcConvB = 1U,
kSimCAdcConvA = 0U,
kSimCAdcConvB = 1U,
kSimCAdcConvA = 0U,
kSimCAdcConvB = 1U }

```

*Defines the type of enumerating ADC converter's ID.*

## SIM HAL driver

- enum `sim_adc_alt_trg_en`  
*SIM ADC alternate trigger enable.*
- enum `sim_dac_hw_trg_sel`  
*DAC0 Hardware Trigger Input Source.*
- enum `sim_ewm_in_src`  
*the ewm\_in source of EWM module.*
- enum `sim_cmp_win_in_src`  
*CMP Sample/Window Input X Source.*
- enum `clock_lptmr_src_t` { ,  
`kClockLptmrSrcMcgIrClk,`  
`kClockLptmrSrcLpoClk,`  
`kClockLptmrSrcEr32kClk,`  
`kClockLptmrSrcOsc0erClkUndiv,`  
`kClockLptmrSrcMcgIrClk,`  
`kClockLptmrSrcLpoClk,`  
`kClockLptmrSrcEr32kClk,`  
`kClockLptmrSrcOsc0erClkUndiv,`  
`kClockLptmrSrcMcgIrClk,`  
`kClockLptmrSrcLpoClk,`  
`kClockLptmrSrcEr32kClk,`  
`kClockLptmrSrcOsc0erClkUndiv,`  
`kClockLptmrSrcMcgIrClk,`  
`kClockLptmrSrcLpoClk,`  
`kClockLptmrSrcEr32kClk,`  
`kClockLptmrSrcOsc0erClkUndiv,`  
`kClockLptmrSrcMcgIrClk,`  
`kClockLptmrSrcLpoClk,`  
`kClockLptmrSrcEr32kClk,`  
`kClockLptmrSrcOsc0erClkUndiv` }  
*LPTMR clock source select.*
- enum `clock_er32k_src_t` { ,  
`kClockEr32kSrcOsc0 = 0U,`  
`kClockEr32kSrcLpo = 3U,`  
`kClockEr32kSrcOsc0 = 0U,`  
`kClockEr32kSrcLpo = 3U,`  
`kClockEr32kSrcOsc0 = 0U,`  
`kClockEr32kSrcLpo = 3U,`  
`kClockEr32kSrcOsc0 = 0U,`  
`kClockEr32kSrcLpo = 3U,`  
`kClockEr32kSrcOsc0 = 0U,`  
`kClockEr32kSrcLpo = 3U` }  
*SIM external reference clock source select (OSC32KSEL).*
- enum `clock_flexcan_src_t` {

```

kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk,
kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk,
kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk,
kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk,
kClockFlexcanSrcOsc0erClk,
kClockFlexcanSrcBusClk }

```

*FLEXCAN clock source select.*

- enum `sim_clock_gate_name_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*
- enum `clock_source_names_t`  
*Clock source and sel names.*
- enum `clock_divider_names_t`  
*Clock Divider names.*
- enum `sim_usbstby_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_flexbus_security_level_t`  
*SIM FlexBus security level.*
- enum `sim_uart_rxsrc_t`  
*SIM UART receive data source select.*
- enum `sim_uart_txsrc_t`  
*SIM UART 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_ch_out_src_t`  
*SIM FlexTimer x channel y output source select.*
- enum `sim_ftm_ft_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.*
- enum `clock_wdog_src_t` {

## SIM HAL driver

- kClockWdogSrcLpoClk,
- kClockWdogSrcAltClk,
- kClockWdogSrcLpoClk,
- kClockWdogSrcAltClk,
- kClockWdogSrcLpoClk,
- kClockWdogSrcAltClk,
- kClockWdogSrcLpoClk,
- kClockWdogSrcAltClk,
- kClockWdogSrcLpoClk,
- kClockWdogSrcAltClk }
- WDOG clock source select.*
- enum `clock_trace_src_t` {
  - kClockTraceSrcMcgoutClk,
  - kClockTraceSrcCoreClk,
  - kClockTraceSrcMcgoutClk,
  - kClockTraceSrcCoreClk,
  - kClockTraceSrcMcgoutClk,
  - kClockTraceSrcCoreClk,
  - kClockTraceSrcMcgoutClk,
  - kClockTraceSrcCoreClk,
  - kClockTraceSrcMcgoutClk,
  - kClockTraceSrcCoreClk }
  - Debug trace clock source select.*
  - enum `clock_nanoedge_clk2x_src` {
    - kClockNanoedgeSrcMcgPllClk,
    - kClockNanoedgeSrcMcgPllClk2x,
    - kClockNanoedgeSrcMcgPllClk,
    - kClockNanoedgeSrcMcgPllClk2x,
    - kClockNanoedgeSrcMcgPllClk,
    - kClockNanoedgeSrcMcgPllClk2x,
    - kClockNanoedgeSrcMcgPllClk,
    - kClockNanoedgeSrcMcgPllClk2x,
    - kClockNanoedgeSrcMcgPllClk,
    - kClockNanoedgeSrcMcgPllClk2x }
    - Debug trace clock source select.*
    - enum `sim_osc32k_clock_sel_t`
      - SIM OSC32KSEL clock source select.*
    - enum `sim_nanoedge_clock_sel_t`
      - SIM NANOEDGECLK2XSEL clock source select.*
    - enum `sim_trace_clock_sel_t`
      - SIM TRACECLKSEL clock source select.*
    - enum `sim_clkout_clock_sel_t`
      - SIM CLKOUT\_SEL clock source select.*
    - enum `sim_adcb_trg_sel_t`
      - SIM ADCB trigger select.*
    - enum `sim_adc_trg_sel_t`
      - SIM ADC trigger select.*

- enum `sim_cadc_conv_id_t` {
  - `kSimCAdcConvA = 0U,`
  - `kSimCAdcConvB = 1U,`
  - `kSimCAdcConvA = 0U,`
  - `kSimCAdcConvB = 1U,`
  - `kSimCAdcConvA = 0U,`
  - `kSimCAdcConvB = 1U,`
  - `kSimCAdcConvA = 0U,`
  - `kSimCAdcConvB = 1U,`
  - `kSimCAdcConvA = 0U,`
  - `kSimCAdcConvB = 1U }`

*Defines the type of enumerating ADC converter's ID.*
- enum `sim_adc_alt_trg_en`

*SIM ADC alternate trigger enable.*
- enum `sim_dac_hw_trg_sel`

*DAC0 Hardware Trigger Input Source.*
- enum `sim_ewm_in_src`

*the ewm\_in source of EWM module.*
- enum `sim_cmp_win_in_src`

*CMP Sample/Window Input X Source.*
- enum `clock_lptmr_src_t` { ,
  - `kClockLptmrSrcMcgIrClk,`
  - `kClockLptmrSrcLpoClk,`
  - `kClockLptmrSrcEr32kClk,`
  - `kClockLptmrSrcOsc0erClkUndiv,`
  - `kClockLptmrSrcMcgIrClk,`
  - `kClockLptmrSrcLpoClk,`
  - `kClockLptmrSrcEr32kClk,`
  - `kClockLptmrSrcOsc0erClkUndiv,`
  - `kClockLptmrSrcMcgIrClk,`
  - `kClockLptmrSrcLpoClk,`
  - `kClockLptmrSrcEr32kClk,`
  - `kClockLptmrSrcOsc0erClkUndiv,`
  - `kClockLptmrSrcMcgIrClk,`
  - `kClockLptmrSrcLpoClk,`
  - `kClockLptmrSrcEr32kClk,`
  - `kClockLptmrSrcOsc0erClkUndiv,`
  - `kClockLptmrSrcMcgIrClk,`
  - `kClockLptmrSrcLpoClk,`
  - `kClockLptmrSrcEr32kClk,`
  - `kClockLptmrSrcOsc0erClkUndiv }`

*LPTMR clock source select.*
- enum `clock_er32k_src_t` { ,

## SIM HAL driver

```

kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U,
kClockEr32kSrcOsc0 = 0U,
kClockEr32kSrcLpo = 3U }

```

*SIM external reference clock source select (OSC32KSEL).*

- enum `clock_flexcan_src_t` {
  - `kClockFlexcanSrcOsc0erClk,`
  - `kClockFlexcanSrcBusClk,`
  - `kClockFlexcanSrcOsc0erClk,`
  - `kClockFlexcanSrcBusClk,`
  - `kClockFlexcanSrcOsc0erClk,`
  - `kClockFlexcanSrcBusClk,`
  - `kClockFlexcanSrcOsc0erClk,`
  - `kClockFlexcanSrcBusClk,`
  - `kClockFlexcanSrcOsc0erClk,`
  - `kClockFlexcanSrcBusClk` }

*FLEXCAN clock source select.*
- enum `sim_clock_gate_name_t`

*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*
- enum `clock_source_names_t`

*Clock source and sel names.*
- enum `clock_divider_names_t`

*Clock Divider names.*
- enum `sim_usbstby_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_flexbus_security_level_t`

*SIM FlexBus security level.*
- enum `sim_uart_rxsrc_t`

*SIM UART receive data source select.*
- enum `sim_uart_txsrc_t`

*SIM UART 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\\_ch\\_out\\_src\\_t](#)
- *SIM FlexTimer x channel y output source select.*  
enum [sim\\_ftm\\_ft\\_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](#) (SIM\_Type \*base, [sim\\_clock\\_gate\\_name\\_t](#) name)  
*Enable the clock for specific module.*
- static void [SIM\\_HAL\\_DisableClock](#) (SIM\_Type \*base, [sim\\_clock\\_gate\\_name\\_t](#) name)  
*Disable the clock for specific module.*
- static bool [SIM\\_HAL\\_GetGateCmd](#) (SIM\_Type \*base, [sim\\_clock\\_gate\\_name\\_t](#) name)  
*Get the the clock gate state for specific module.*
- static void [CLOCK\\_HAL\\_SetExternalRefClock32kSrc](#) (SIM\_Type \*base, [clock\\_er32k\\_src\\_t](#) setting)  
*Set the clock selection of ERCLK32K.*
- static [clock\\_er32k\\_src\\_t](#) [CLOCK\\_HAL\\_GetExternalRefClock32kSrc](#) (SIM\_Type \*base)  
*Get the clock selection of ERCLK32K.*
- static void [CLOCK\\_HAL\\_SetOsc32kOutSel](#) (SIM\_Type \*base, [clock\\_osc32kout\\_sel\\_t](#) setting)  
*Set OSC32KOUT selection.*
- static [clock\\_osc32kout\\_sel\\_t](#) [CLOCK\\_HAL\\_GetOsc32kOutSel](#) (SIM\_Type \*base)  
*Get OSC32KOUT selection.*
- static uint32\_t [SIM\\_HAL\\_GetRamSize](#) (SIM\_Type \*base)  
*Gets RAM size.*
- static void [CLOCK\\_HAL\\_SetPlflSel](#) (SIM\_Type \*base, [clock\\_pllfl\\_sel\\_t](#) setting)  
*Set PLL/FLL clock selection.*
- static [clock\\_pllfl\\_sel\\_t](#) [CLOCK\\_HAL\\_GetPlflSel](#) (SIM\_Type \*base)  
*Get PLL/FLL clock selection.*
- static void [CLOCK\\_HAL\\_SetTraceClkSrc](#) (SIM\_Type \*base, [clock\\_trace\\_src\\_t](#) setting)  
*Set debug trace clock selection.*
- static [clock\\_trace\\_src\\_t](#) [CLOCK\\_HAL\\_GetTraceClkSrc](#) (SIM\_Type \*base)  
*Get debug trace clock selection.*
- static void [CLOCK\\_HAL\\_SetClkOutSel](#) (SIM\_Type \*base, [clock\\_clkout\\_src\\_t](#) setting)  
*Set CLKOUTSEL selection.*
- static [clock\\_clkout\\_src\\_t](#) [CLOCK\\_HAL\\_GetClkOutSel](#) (SIM\_Type \*base)  
*Get CLKOUTSEL selection.*
- static void [CLOCK\\_HAL\\_SetOutDiv1](#) (SIM\_Type \*base, uint8\_t setting)  
*Set OUTDIV1.*
- static uint8\_t [CLOCK\\_HAL\\_GetOutDiv1](#) (SIM\_Type \*base)  
*Get OUTDIV1.*
- static void [CLOCK\\_HAL\\_SetOutDiv2](#) (SIM\_Type \*base, uint8\_t setting)  
*Set OUTDIV2.*
- static uint8\_t [CLOCK\\_HAL\\_GetOutDiv2](#) (SIM\_Type \*base)  
*Get OUTDIV2.*
- static void [CLOCK\\_HAL\\_SetOutDiv4](#) (SIM\_Type \*base, uint8\_t setting)

## SIM HAL driver

- Set OUTDIV4.*

  - static uint8\_t **CLOCK\_HAL\_GetOutDiv4** (SIM\_Type \*base)

*Get OUTDIV4.*

  - void **SIM\_HAL\_SetFtmTriggerSrcMode** (SIM\_Type \*base, 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** (SIM\_Type \*base, uint32\_t instance, uint8\_t trigger)

*Gets the FlexTimer x hardware trigger y source select setting.*

  - void **SIM\_HAL\_SetFtmExternalClkPinMode** (SIM\_Type \*base, 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** (SIM\_Type \*base, uint32\_t instance)

*Gets the FlexTimer x external clock pin select setting.*

  - void **SIM\_HAL\_SetFtmChSrcMode** (SIM\_Type \*base, 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** (SIM\_Type \*base, uint32\_t instance, uint8\_t channel)

*Gets the FlexTimer x channel y input capture source select setting.*

  - void **SIM\_HAL\_SetFtmFaultSelMode** (SIM\_Type \*base, uint32\_t instance, uint8\_t fault, **sim\_ftm\_ft\_sel\_t** select)

*Sets the FlexTimer x fault y select setting.*

  - **sim\_ftm\_ft\_sel\_t** **SIM\_HAL\_GetFtmFaultSelMode** (SIM\_Type \*base, uint32\_t instance, uint8\_t fault)

*Gets the FlexTimer x fault y select setting.*

  - void **SIM\_HAL\_SetFtmChOutSrcMode** (SIM\_Type \*base, 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** (SIM\_Type \*base, uint32\_t instance, uint8\_t channel)

*Gets the FlexTimer x channel y output source select setting.*

  - void **SIM\_HAL\_SetFtmSyncCmd** (SIM\_Type \*base, uint32\_t instance, bool sync)

*Set FlexTimer x hardware trigger 0 software synchronization.*

  - static bool **SIM\_HAL\_GetFtmSyncCmd** (SIM\_Type \*base, uint32\_t instance)

*Get FlexTimer x hardware trigger 0 software synchronization setting.*

  - static uint32\_t **SIM\_HAL\_GetFamilyId** (SIM\_Type \*base)

*Gets the Kinetis Family ID in the System Device ID register (SIM\_SDID).*

  - static uint32\_t **SIM\_HAL\_GetSubFamilyId** (SIM\_Type \*base)

*Gets the Kinetis Sub-Family ID in the System Device ID register (SIM\_SDID).*

  - static uint32\_t **SIM\_HAL\_GetSeriesId** (SIM\_Type \*base)

*Gets the Kinetis SeriesID in the System Device ID register (SIM\_SDID).*

  - static uint32\_t **SIM\_HAL\_GetRevId** (SIM\_Type \*base)

*Gets the Kinetis Revision ID in the System Device ID register (SIM\_SDID).*

  - static uint32\_t **SIM\_HAL\_GetDieId** (SIM\_Type \*base)

*Gets the Kinetis Die ID in the System Device ID register (SIM\_SDID).*

  - static uint32\_t **SIM\_HAL\_GetFamId** (SIM\_Type \*base)

*Gets the Kinetis family identification in the System Device ID register (SIM\_SDID).*

  - static uint32\_t **SIM\_HAL\_GetPinCntId** (SIM\_Type \*base)

*Gets the Kinetis Pincount ID in System Device ID register (SIM\_SDID).*



- static uint32\_t [SIM\\_HAL\\_GetProgramFlashSize](#) (SIM\_Type \*base)  
*Gets the program flash size in the Flash Configuration Register 1 (SIM\_FCFG).*
- static void [SIM\\_HAL\\_SetFlashDoze](#) (SIM\_Type \*base, uint32\_t setting)  
*Sets the Flash Doze in the Flash Configuration Register 1 (SIM\_FCFG).*
- static uint32\_t [SIM\\_HAL\\_GetFlashDoze](#) (SIM\_Type \*base)  
*Gets the Flash Doze in the Flash Configuration Register 1 (SIM\_FCFG).*
- static void [SIM\\_HAL\\_SetFlashDisableCmd](#) (SIM\_Type \*base, bool disable)  
*Sets the Flash disable setting.*
- static bool [SIM\\_HAL\\_GetFlashDisableCmd](#) (SIM\_Type \*base)  
*Gets the Flash disable setting.*
- static uint32\_t [SIM\\_HAL\\_GetFlashMaxAddrBlock0](#) (SIM\_Type \*base)  
*Gets the Flash maximum address block 0 in the Flash Configuration Register 1 (SIM\_FCFG).*
- static void [CLOCK\\_HAL\\_SetSdhcSrc](#) (SIM\_Type \*base, uint32\_t instance, clock\_sdhc\_src\_t setting)  
*Set the SDHC clock source selection.*
- static clock\_sdhc\_src\_t [CLOCK\\_HAL\\_GetSdhcSrc](#) (SIM\_Type \*base, uint32\_t instance)  
*Get the SDHC clock source selection.*
- static void [CLOCK\\_HAL\\_SetTimeSrc](#) (SIM\_Type \*base, uint32\_t instance, clock\_time\_src\_t setting)  
*Set the ethernet timestamp clock source selection.*
- static clock\_time\_src\_t [CLOCK\\_HAL\\_GetTimeSrc](#) (SIM\_Type \*base, uint32\_t instance)  
*Get the ethernet timestamp clock source selection.*
- static void [CLOCK\\_HAL\\_SetRmiiSrc](#) (SIM\_Type \*base, uint32\_t instance, clock\_rmii\_src\_t setting)  
*Set the Ethernet RMII interface clock source selection.*
- static clock\_rmii\_src\_t [CLOCK\\_HAL\\_GetRmiiSrc](#) (SIM\_Type \*base, uint32\_t instance)  
*Get the Ethernet RMII interface clock source selection.*
- static void [CLOCK\\_HAL\\_SetRtcClkOutSel](#) (SIM\_Type \*base, clock\_rtcout\_src\_t setting)  
*Set RTCCLKOUTSEL selection.*
- static clock\_rtcout\_src\_t [CLOCK\\_HAL\\_GetRtcClkOutSel](#) (SIM\_Type \*base)  
*Get RTCCLKOUTSEL selection.*
- static void [CLOCK\\_HAL\\_SetOutDiv3](#) (SIM\_Type \*base, uint8\_t setting)  
*Set OUTDIV3.*
- static uint8\_t [CLOCK\\_HAL\\_GetOutDiv3](#) (SIM\_Type \*base)  
*Get OUTDIV3.*
- static void [SIM\\_HAL\\_SetPtd7PadDriveStrengthMode](#) (SIM\_Type \*base, sim\_ptd7pad\_strenght\_t setting)  
*Sets the PTD7 pad drive strength setting.*
- static sim\_ptd7pad\_strenght\_t [SIM\\_HAL\\_GetPtd7PadDriveStrengthMode](#) (SIM\_Type \*base)  
*Gets the PTD7 pad drive strength setting.*
- static void [SIM\\_HAL\\_SetFlexbusSecurityLevelMode](#) (SIM\_Type \*base, sim\_flexbus\_security\_level\_t setting)  
*Sets the FlexBus security level setting.*
- static sim\_flexbus\_security\_level\_t [SIM\\_HAL\\_GetFlexbusSecurityLevelMode](#) (SIM\_Type \*base)  
*Gets the FlexBus security level setting.*
- static uint32\_t [SIM\\_HAL\\_GetFlexnvmSize](#) (SIM\_Type \*base)  
*Gets the FlexNVM size in the Flash Configuration Register 1 (SIM\_FCFG).*
- static uint32\_t [SIM\\_HAL\\_GetEepromSize](#) (SIM\_Type \*base)  
*Gets the EEPROM size in the Flash Configuration Register 1 (SIM\_FCFG).*
- static uint32\_t [SIM\\_HAL\\_GetFlexnvmPartition](#) (SIM\_Type \*base)  
*Gets the FlexNVM partition in the Flash Configuration Register 1 (SIM\_FCFG).*

## SIM HAL driver

- static uint32\_t [SIM\\_HAL\\_GetFlashMaxAddrBlock1](#) (SIM\_Type \*base)  
*Gets the Flash maximum address block 1 in Flash Configuration Register 2.*
- static uint32\_t [SIM\\_HAL\\_GetProgramFlashCmd](#) (SIM\_Type \*base)  
*Gets the program flash in the Flash Configuration Register 2.*
- static bool [SIM\\_HAL\\_GetSwapProgramFlash](#) (SIM\_Type \*base)  
*Gets the Swap program flash flag in the Flash Configuration Register 2.*
- void [CLOCK\\_HAL\\_SetUsbfsDiv](#) (SIM\_Type \*base, uint8\_t usbdiv, uint8\_t usbfrac)  
*Set USB FS divider setting.*
- void [CLOCK\\_HAL\\_GetUsbfsDiv](#) (SIM\_Type \*base, uint8\_t \*usbdiv, uint8\_t \*usbfrac)  
*Get USB FS divider setting.*
- static void [CLOCK\\_HAL\\_SetUsbfsSrc](#) (SIM\_Type \*base, 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](#) (SIM\_Type \*base, uint32\_t instance)  
*Get the selection of the clock source for the USB FS 48 MHz clock.*
- static void [SIM\\_HAL\\_SetUsbVoltRegulatorCmd](#) (SIM\_Type \*base, bool enable)  
*Sets the USB voltage regulator enabled setting.*
- static bool [SIM\\_HAL\\_GetUsbVoltRegulatorCmd](#) (SIM\_Type \*base)  
*Gets the USB voltage regulator enabled setting.*
- static void [SIM\\_HAL\\_SetUsbVoltRegulatorInStdbyDuringStopMode](#) (SIM\_Type \*base, 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](#) (SIM\_Type \*base)  
*Gets the USB voltage regulator in a standby mode setting.*
- static void [SIM\\_HAL\\_SetUsbVoltRegulatorInStdbyDuringVlprwMode](#) (SIM\_Type \*base, 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](#) (SIM\_Type \*base)  
*Gets the USB voltage regulator in a standby mode during the VLPR or the VLPW.*
- static void [SIM\\_HAL\\_SetUsbVoltRegulatorInStdbyDuringStopCmd](#) (SIM\_Type \*base, bool enable)  
*Sets the USB voltage regulator stop standby write enable setting.*
- static bool [SIM\\_HAL\\_GetUsbVoltRegulatorInStdbyDuringStopCmd](#) (SIM\_Type \*base)  
*Gets the USB voltage regulator stop standby write enable setting.*
- static void [SIM\\_HAL\\_SetUsbVoltRegulatorInStdbyDuringVlprwCmd](#) (SIM\_Type \*base, bool enable)  
*Sets the USB voltage regulator VLP standby write enable setting.*
- static bool [SIM\\_HAL\\_GetUsbVoltRegulatorInStdbyDuringVlprwCmd](#) (SIM\_Type \*base)  
*Gets the USB voltage regulator VLP standby write enable setting.*
- static void [SIM\\_HAL\\_SetUsbVoltRegulatorWriteCmd](#) (SIM\_Type \*base, bool enable)  
*Sets the USB voltage regulator enable write enable setting.*
- static bool [SIM\\_HAL\\_GetUsbVoltRegulatorWriteCmd](#) (SIM\_Type \*base)  
*Gets the USB voltage regulator enable write enable setting.*
- static void [CLOCK\\_HAL\\_SetLpuartSrc](#) (SIM\_Type \*base, uint32\_t instance, clock\_lpuart\_src\_t setting)  
*Set LPUART clock source.*
- static clock\_lpuart\_src\_t [CLOCK\\_HAL\\_GetLpuartSrc](#) (SIM\_Type \*base, uint32\_t instance)  
*Get LPUART clock source.*

- static void [SIM\\_HAL\\_SetLpuartRxSrcMode](#) (SIM\_Type \*base, uint32\_t instance, sim\_lpuart\_rxsrc\_t select)
  - Sets the LPUARTx receive data source select setting.*
- static sim\_lpuart\_rxsrc\_t [SIM\\_HAL\\_GetLpuartRxSrcMode](#) (SIM\_Type \*base, uint32\_t instance)
  - Gets the LPUARTx receive data source select setting.*
- static void [CLOCK\\_HAL\\_SetUsbhsSlowClockSrc](#) (SIM\_Type \*base, uint32\_t instance, clock\_usbhs\_slowclk\_src\_t setting)
  - Set the selection of the clock source for the USB HS/USB PHY slow clock.*
- static clock\_usbhs\_slowclk\_src\_t [CLOCK\\_HAL\\_GetUsbhsSlowClockSrc](#) (SIM\_Type \*base, uint32\_t instance)
  - Get the selection of the clock source for the USB HS/USB PHY slow clock.*
- void [CLOCK\\_HAL\\_SetPIIFllDiv](#) (SIM\_Type \*base, uint8\_t pllflldiv, uint8\_t pllflfrac)
  - Set PLL/FLL divider setting.*
- void [CLOCK\\_HAL\\_GetPIIFllDiv](#) (SIM\_Type \*base, uint8\_t \*pllflldiv, uint8\_t \*pllflfrac)
  - Gets PLL/FLL divider setting.*
- void [CLOCK\\_HAL\\_SetTraceDiv](#) (SIM\_Type \*base, uint8\_t tracediv, uint8\_t tracefrac)
  - Set TRACECLK divider setting.*
- void [CLOCK\\_HAL\\_GetTraceDiv](#) (SIM\_Type \*base, uint8\_t \*tracediv, uint8\_t \*tracefrac)
  - Gets TRACECLK setting.*
- static void [CLOCK\\_HAL\\_SetTpmSrc](#) (SIM\_Type \*base, uint32\_t instance, clock\_tpm\_src\_t setting)
  - Set the TPM clock source selection.*
- static clock\_tpm\_src\_t [CLOCK\\_HAL\\_GetTpmSrc](#) (SIM\_Type \*base, uint32\_t instance)
  - Get the TPM clock source selection.*
- static void [SIM\\_HAL\\_SetUsbVoltRegulatorInrushLimitCmd](#) (SIM\_Type \*base, bool enable)
  - Sets the USB voltage regulator inrush current limit setting.*
- static bool [SIM\\_HAL\\_GetUsbVoltRegulatorInrushLimitCmd](#) (SIM\_Type \*base)
  - Gets the USB voltage regulator inrush current limit setting.*
- static void [SIM\\_HAL\\_SetUsbVoltRegulatorOutputTargetCmd](#) (SIM\_Type \*base, sim\_usbvout\_mode\_t target)
  - Sets the USB voltage regulator output target.*
- static sim\_usbvout\_mode\_t [SIM\\_HAL\\_GetUsbVoltRegulatorOutputTargetCmd](#) (SIM\_Type \*base)
  - Gets the USB voltage regulator output target.*
- static void [SIM\\_HAL\\_SetUsbPhyPllRegulatorCmd](#) (SIM\_Type \*base, bool enable)
  - Sets the USB PHY PLL regulator enabled setting.*
- static bool [SIM\\_HAL\\_GetUsbPhyPllRegulatorCmd](#) (SIM\_Type \*base)
  - Gets the USB PHY PLL regulator enabled setting.*
- static void [SIM\\_HAL\\_SetLpuartTxSrcMode](#) (SIM\_Type \*base, uint32\_t instance, sim\_lpuart\_txsrc\_t select)
  - Sets the LPUARTx transmit data source select setting.*
- static sim\_lpuart\_txsrc\_t [SIM\\_HAL\\_GetLpuartTxSrcMode](#) (SIM\_Type \*base, uint32\_t instance)
  - Gets the LPUARTx transmit data source select setting.*
- static void [CLOCK\\_HAL\\_SetLpsciSrc](#) (SIM\_Type \*base, uint32\_t instance, clock\_lpsci\_src\_t setting)
  - Set the LPSCI clock source selection.*
- static clock\_lpsci\_src\_t [CLOCK\\_HAL\\_GetLpsciSrc](#) (SIM\_Type \*base, uint32\_t instance)
  - Get the LPSCI clock source selection.*
- static void [SIM\\_HAL\\_SetLpsciRxSrcMode](#) (SIM\_Type \*base, uint32\_t instance, sim\_lpsci\_rxsrc\_t select)
  - Sets the LPSCIx receive data source select setting.*

## SIM HAL driver

- static `sim_lpsci_rxsrc_t SIM_HAL_GetLpsciRxSrcMode` (SIM\_Type \*base, uint32\_t instance)  
*Gets the LPSCIx receive data source select setting.*
- static void `SIM_HAL_SetLpsciTxSrcMode` (SIM\_Type \*base, uint32\_t instance, `sim_lpsci_txsrc_t` select)  
*Sets the LPSCIx transmit data source select setting.*
- static `sim_lpsci_txsrc_t SIM_HAL_GetLpsciTxSrcMode` (SIM\_Type \*base, uint32\_t instance)  
*Gets the LPSCIx transmit data source select setting.*
- static uint32\_t `SIM_HAL_GetSramSize` (SIM\_Type \*base)  
*Gets the Kinetis SramSize in the System Device ID register (SIM\_SDID).*
- static void `CLOCK_HAL_SetCopSrc` (SIM\_Type \*base, `clock_cop_src_t` setting)  
*Set the clock selection of COP.*
- static `clock_cop_src_t CLOCK_HAL_GetCopSrc` (SIM\_Type \*base)  
*Get the clock selection of COP.*
- static void `SIM_HAL_SetLpuartOpenDrainCmd` (SIM\_Type \*base, uint32\_t instance, bool enable)  
*Sets the LPUARTx Open Drain Enable setting.*
- static bool `SIM_HAL_GetLpuartOpenDrainCmd` (SIM\_Type \*base, uint32\_t instance)  
*Gets the LPUARTx Open Drain Enable setting.*
- static void `SIM_HAL_SetTpmChSrcMode` (SIM\_Type \*base, 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` (SIM\_Type \*base, uint32\_t instance, uint8\_t channel)  
*Gets the Timer/PWM x channel y input capture source select setting.*
- void `SIM_HAL_SetTpmExternalClkPinSelMode` (SIM\_Type \*base, 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` (SIM\_Type \*base, uint32\_t instance)  
*Gets the Timer/PWM x external clock pin select setting.*
- static void `CLOCK_HAL_SetFlexioSrc` (SIM\_Type \*base, uint32\_t instance, `clock_flexio_src_-t` setting)  
*Select the clock source for FLEXIO.*
- static `clock_flexio_src_t CLOCK_HAL_GetFlexioSrc` (SIM\_Type \*base, uint32\_t instance)  
*Get the clock source of FLEXIO.*
- static void `SIM_HAL_SetUartOpenDrainCmd` (SIM\_Type \*base, uint32\_t instance, bool enable)  
*Sets the UARTx Open Drain Enable setting.*
- static bool `SIM_HAL_GetUartOpenDrainCmd` (SIM\_Type \*base, uint32\_t instance)  
*Gets the UARTx Open Drain Enable setting.*
- static uint32\_t `SIM_HAL_GetSramSizeId` (SIM\_Type \*base)  
*Gets the Kinetis SRAMSIZE ID in the System Device ID register (SIM\_SDID).*
- static void `CLOCK_HAL_SetOutDiv5ENCmd` (SIM\_Type \*base, bool setting)  
*Set OUTDIV5EN.*
- static bool `CLOCK_HAL_GetOutDiv5ENCmd` (SIM\_Type \*base)  
*Get OUTDIV5EN.*
- static void `CLOCK_HAL_SetOutDiv5` (SIM\_Type \*base, uint8\_t setting)  
*Set OUTDIV5.*
- static uint8\_t `CLOCK_HAL_GetOutDiv5` (SIM\_Type \*base)  
*Get OUTDIV5.*
- void `CLOCK_HAL_SetAdcAltClkSrc` (SIM\_Type \*base, 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](#) (SIM\_Type \*base, uint32\_t instance)

*Gets the ADC ALT clock source selection setting.*

  - `void` [SIM\\_HAL\\_SetUartOpenDrainMode](#) (SIM\_Type \*base, uint32\_t instance, bool enable)

*Sets the UARTx open drain enable setting.*

  - `bool` [SIM\\_HAL\\_GetUartOpenDrainMode](#) (SIM\_Type \*base, uint32\_t instance)

*Gets the UARTx open drain enable setting.*

  - `static void` [CLOCK\\_HAL\\_SetFtmFixFreqClkSrc](#) (SIM\_Type \*base, clock\_ftm\_fixedfreq\_src\_t ftmFixedFreqSel)

*Sets the FTM Fixed clock source selection setting.*

  - `static clock_ftm_fixedfreq_src_t` [CLOCK\\_HAL\\_GetFtmFixFreqClkSrc](#) (SIM\_Type \*base)

*Gets the FTM Fixed clock source selection setting.*

  - `static void` [SIM\\_HAL\\_SetFtmCarrierFreqMode](#) (SIM\_Type \*base, sim\_ftm\_ftt\_carrier\_sel\_t select)

*Sets the Carrier frequency selection for FTM0/2 output channel.*

  - `static sim_ftm_ftt_carrier_sel_t` [SIM\\_HAL\\_GetFtmCarrierFreqMode](#) (SIM\_Type \*base)

*Gets the Carrier frequency selection for FTM0/2 output channel.*

  - `static uint32_t` [SIM\\_HAL\\_GetSubFamId](#) (SIM\_Type \*base)

*Gets the Kinetis SbuFam ID in System Device ID register (SIM\_SDID).*

  - `static uint32_t` [SIM\\_HAL\\_GetFlashMaxAddrBlock](#) (SIM\_Type \*base)

*Gets the Flash maximum address block in the Flash Configuration Register 1 (SIM\_FCFG).*

  - `sim_hal_status_t` [CLOCK\\_HAL\\_SetSource](#) (SIM\_Type \*base, clock\_source\_names\_t clockSource, uint8\_t setting)

*Sets the clock source setting.*

  - `sim_hal_status_t` [CLOCK\\_HAL\\_GetSource](#) (SIM\_Type \*base, clock\_source\_names\_t clockSource, uint8\_t \*setting)

*Gets the clock source setting.*

  - `sim_hal_status_t` [CLOCK\\_HAL\\_SetDivider](#) (SIM\_Type \*base, clock\_divider\_names\_t clockDivider, uint32\_t setting)

*Sets the clock divider setting.*

## IP related clock feature APIs

- `void` [CLOCK\\_HAL\\_SetOutDiv](#) (SIM\_Type \*base, uint8\_t outdiv1, uint8\_t outdiv2, uint8\_t outdiv3, uint8\_t outdiv4)

*Sets the clock out dividers setting.*

  - `void` [CLOCK\\_HAL\\_GetOutDiv](#) (SIM\_Type \*base, uint8\_t \*outdiv1, uint8\_t \*outdiv2, uint8\_t \*outdiv3, uint8\_t \*outdiv4)

*Gets the clock out dividers setting.*

  - `void` [SIM\\_HAL\\_SetAdcAlternativeTriggerCmd](#) (SIM\_Type \*base, uint32\_t instance, bool enable)

*Sets the ADCx alternate trigger enable setting.*

  - `bool` [SIM\\_HAL\\_GetAdcAlternativeTriggerCmd](#) (SIM\_Type \*base, uint32\_t instance)

*Gets the ADCx alternate trigger enable setting.*

  - `void` [SIM\\_HAL\\_SetAdcPreTriggerMode](#) (SIM\_Type \*base, uint32\_t instance, sim\_adc\_pretrg\_sel\_t select)

*Sets the ADCx pre-trigger select setting.*

  - `sim_adc_pretrg_sel_t` [SIM\\_HAL\\_GetAdcPreTriggerMode](#) (SIM\_Type \*base, uint32\_t instance)

*Gets the ADCx pre-trigger select setting.*

  - `void` [SIM\\_HAL\\_SetAdcTriggerMode](#) (SIM\_Type \*base, uint32\_t instance, sim\_adc\_trg\_sel\_t select)

## SIM HAL driver

- Sets the ADCx trigger select setting.*
- `sim_adc_trg_sel_t SIM_HAL_GetAdcTriggerMode` (SIM\_Type \*base, uint32\_t instance)
- Gets the ADCx trigger select setting.*
- void `SIM_HAL_SetAdcTriggerModeOneStep` (SIM\_Type \*base, 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` (SIM\_Type \*base, uint32\_t instance, `sim_uart_rxsrc_t` select)
- Sets the UARTx receive data source select setting.*
- `sim_uart_rxsrc_t SIM_HAL_GetUartRxSrcMode` (SIM\_Type \*base, uint32\_t instance)
- Gets the UARTx receive data source select setting.*
- void `SIM_HAL_SetUartTxSrcMode` (SIM\_Type \*base, uint32\_t instance, `sim_uart_txsrc_t` select)
- Sets the UARTx transmit data source select setting.*
- `sim_uart_txsrc_t SIM_HAL_GetUartTxSrcMode` (SIM\_Type \*base, uint32\_t instance)
- Gets the UARTx transmit data source select setting.*

### 53.2.5 Macro Definition Documentation

53.2.5.1 #define FSL\_SIM\_SCGC\_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)

53.2.5.2 #define FSL\_SIM\_SCGC\_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)

53.2.5.3 #define FSL\_SIM\_SCGC\_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)

53.2.5.4 #define FSL\_SIM\_SCGC\_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)

53.2.5.5 #define FSL\_SIM\_SCGC\_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)

### 53.2.6 Enumeration Type Documentation

#### 53.2.6.1 enum sim\_hal\_status\_t

Enumerator

*kSimHalSuccess* Success.  
*kSimHalFail* Error occurs.

#### 53.2.6.2 enum clock\_lptmr\_src\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.

*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 53.2.6.3 enum clock\_pllfl\_sel\_t

SIM PLLFLLSEL clock source select

### 53.2.6.4 enum clock\_er32k\_src\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.6.5 enum sim\_adc\_pretrg\_sel\_t

SIM ADCx pre-trigger select

## SIM HAL driver

### 53.2.6.6 enum clock\_lptmr\_src\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 53.2.6.7 enum clock\_pllfl\_sel\_t

SIM PLLFLLSEL clock source select

### 53.2.6.8 enum clock\_er32k\_src\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.



### 53.2.6.9 enum sim\_adc\_pretrg\_sel\_t

SIM ADCx pre-trigger select

### 53.2.6.10 enum clock\_lptmr\_src\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 53.2.6.11 enum clock\_pllfl\_sel\_t

SIM PLLFLLSEL clock source select

### 53.2.6.12 enum clock\_er32k\_src\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).

## SIM HAL driver

*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.6.13 enum sim\_adc\_pretrg\_sel\_t

SIM ADCx pre-trigger select

### 53.2.6.14 enum clock\_lptmr\_src\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 53.2.6.15 enum clock\_pllfl\_sel\_t

SIM PLLFLLSEL clock source select

### 53.2.6.16 enum clock\_er32k\_src\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).

*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.6.17 enum sim\_adc\_pretrg\_sel\_t

SIM ADCx pre-trigger select

### 53.2.6.18 enum clock\_wdog\_src\_t

Enumerator

*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.

### 53.2.6.19 enum clock\_trace\_src\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock

## SIM HAL driver

### 53.2.6.20 enum clock\_nanoedge\_clk2x\_src

Enumerator

*kClockNanoedgeSrcMcgPl1Clk* MCG out clock.  
*kClockNanoedgeSrcMcgPl1Clk2x* core clock  
*kClockNanoedgeSrcMcgPl1Clk* MCG out clock.  
*kClockNanoedgeSrcMcgPl1Clk2x* core clock  
*kClockNanoedgeSrcMcgPl1Clk* MCG out clock.  
*kClockNanoedgeSrcMcgPl1Clk2x* core clock  
*kClockNanoedgeSrcMcgPl1Clk* MCG out clock.  
*kClockNanoedgeSrcMcgPl1Clk2x* core clock  
*kClockNanoedgeSrcMcgPl1Clk* MCG out clock.  
*kClockNanoedgeSrcMcgPl1Clk2x* core clock

### 53.2.6.21 enum sim\_cadc\_conv\_id\_t

Enumerator

*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.

### 53.2.6.22 enum sim\_ewm\_in\_src

### 53.2.6.23 enum clock\_lptmr\_src\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

#### 53.2.6.24 enum clock\_er32k\_src\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.

#### 53.2.6.25 enum clock\_flexcan\_src\_t

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.

## SIM HAL driver

### 53.2.6.26 enum sim\_clock\_gate\_name\_t

### 53.2.6.27 enum clock\_wdog\_src\_t

Enumerator

*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.

### 53.2.6.28 enum clock\_trace\_src\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock

### 53.2.6.29 enum clock\_nanoedge\_clk2x\_src

Enumerator

*kClockNanoedgeSrcMcgPliClk* MCG out clock.  
*kClockNanoedgeSrcMcgPliClk2x* core clock  
*kClockNanoedgeSrcMcgPliClk* MCG out clock.  
*kClockNanoedgeSrcMcgPliClk2x* core clock  
*kClockNanoedgeSrcMcgPliClk* MCG out clock.  
*kClockNanoedgeSrcMcgPliClk2x* core clock  
*kClockNanoedgeSrcMcgPliClk* MCG out clock.  
*kClockNanoedgeSrcMcgPliClk2x* core clock

*kClockNanoedgeSrcMcgPlIClk* MCG out clock.  
*kClockNanoedgeSrcMcgPlIClk2x* core clock

### 53.2.6.30 enum sim\_cadc\_conv\_id\_t

Enumerator

*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.

### 53.2.6.31 enum sim\_ewm\_in\_src

### 53.2.6.32 enum clock\_lptmr\_src\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

## SIM HAL driver

### 53.2.6.33 enum clock\_er32k\_src\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.6.34 enum clock\_flexcan\_src\_t

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.

### 53.2.6.35 enum sim\_clock\_gate\_name\_t

### 53.2.6.36 enum clock\_wdog\_src\_t

Enumerator

*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.



*kClockWdogSrcLpoClk* LPO.

*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.

### 53.2.6.37 enum clock\_trace\_src\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

### 53.2.6.38 enum clock\_nanoedge\_clk2x\_src

Enumerator

*kClockNanoedgeSrcMcgPllClk* MCG out clock.

*kClockNanoedgeSrcMcgPllClk2x* core clock

*kClockNanoedgeSrcMcgPllClk* MCG out clock.

*kClockNanoedgeSrcMcgPllClk2x* core clock

*kClockNanoedgeSrcMcgPllClk* MCG out clock.

*kClockNanoedgeSrcMcgPllClk2x* core clock

*kClockNanoedgeSrcMcgPllClk* MCG out clock.

*kClockNanoedgeSrcMcgPllClk2x* core clock

*kClockNanoedgeSrcMcgPllClk* MCG out clock.

*kClockNanoedgeSrcMcgPllClk2x* core clock

### 53.2.6.39 enum sim\_cadc\_conv\_id\_t

Enumerator

*kSimCAdcConvA* ID for ADC converter A.

*kSimCAdcConvB* ID for ADC converter B.

*kSimCAdcConvA* ID for ADC converter A.

*kSimCAdcConvB* ID for ADC converter B.

*kSimCAdcConvA* ID for ADC converter A.

*kSimCAdcConvB* ID for ADC converter B.

## SIM HAL driver

*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.

### 53.2.6.40 enum sim\_ewm\_in\_src

### 53.2.6.41 enum clock\_lptmr\_src\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 53.2.6.42 enum clock\_er32k\_src\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).

*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.6.43 enum clock\_flexcan\_src\_t

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.

### 53.2.6.44 enum sim\_clock\_gate\_name\_t

### 53.2.6.45 enum clock\_wdog\_src\_t

Enumerator

*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.

### 53.2.6.46 enum clock\_trace\_src\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.

## SIM HAL driver

*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock

### 53.2.6.47 enum clock\_nanoedge\_clk2x\_src

Enumerator

*kClockNanoedgeSrcMcgPlIClk* MCG out clock.  
*kClockNanoedgeSrcMcgPlIClk2x* core clock  
*kClockNanoedgeSrcMcgPlIClk* MCG out clock.  
*kClockNanoedgeSrcMcgPlIClk2x* core clock  
*kClockNanoedgeSrcMcgPlIClk* MCG out clock.  
*kClockNanoedgeSrcMcgPlIClk2x* core clock  
*kClockNanoedgeSrcMcgPlIClk* MCG out clock.  
*kClockNanoedgeSrcMcgPlIClk2x* core clock  
*kClockNanoedgeSrcMcgPlIClk* MCG out clock.  
*kClockNanoedgeSrcMcgPlIClk2x* core clock

### 53.2.6.48 enum sim\_cadc\_conv\_id\_t

Enumerator

*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.

### 53.2.6.49 enum sim\_ewm\_in\_src

### 53.2.6.50 enum clock\_lptmr\_src\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 53.2.6.51 enum clock\_er32k\_src\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.

## SIM HAL driver

### 53.2.6.52 enum clock\_flexcan\_src\_t

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.

### 53.2.6.53 enum sim\_clock\_gate\_name\_t

### 53.2.6.54 enum clock\_wdog\_src\_t

Enumerator

*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.  
*kClockWdogSrcLpoClk* LPO.  
*kClockWdogSrcAltClk* Alternative clock, for K64F12 it is Bus clock.

### 53.2.6.55 enum clock\_trace\_src\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock  
*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock

*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock

### 53.2.6.56 enum clock\_nanoedge\_clk2x\_src

Enumerator

*kClockNanoedgeSrcMcgPlkClk* MCG out clock.  
*kClockNanoedgeSrcMcgPlkClk2x* core clock  
*kClockNanoedgeSrcMcgPlkClk* MCG out clock.  
*kClockNanoedgeSrcMcgPlkClk2x* core clock  
*kClockNanoedgeSrcMcgPlkClk* MCG out clock.  
*kClockNanoedgeSrcMcgPlkClk2x* core clock  
*kClockNanoedgeSrcMcgPlkClk* MCG out clock.  
*kClockNanoedgeSrcMcgPlkClk2x* core clock  
*kClockNanoedgeSrcMcgPlkClk* MCG out clock.  
*kClockNanoedgeSrcMcgPlkClk2x* core clock

### 53.2.6.57 enum sim\_cadc\_conv\_id\_t

Enumerator

*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.  
*kSimCAdcConvA* ID for ADC converter A.  
*kSimCAdcConvB* ID for ADC converter B.

### 53.2.6.58 enum sim\_ewm\_in\_src

### 53.2.6.59 enum clock\_lptmr\_src\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

## SIM HAL driver

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.  
*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 53.2.6.60 enum clock\_er32k\_src\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.  
*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.6.61 enum clock\_flexcan\_src\_t

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.  
*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.



*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.

### 53.2.6.62 enum sim\_clock\_gate\_name\_t

## 53.2.7 Function Documentation

### 53.2.7.1 static void SIM\_HAL\_EnableClock ( SIM\_Type \* *base*, sim\_clock\_gate\_name\_t *name* ) [inline], [static]

This function enables the clock for specific module.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
| <i>name</i> | Name of the module to enable.          |

### 53.2.7.2 static void SIM\_HAL\_DisableClock ( SIM\_Type \* *base*, sim\_clock\_gate\_name\_t *name* ) [inline], [static]

This function disables the clock for specific module.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
| <i>name</i> | Name of the module to disable.         |

### 53.2.7.3 static bool SIM\_HAL\_GetGateCmd ( SIM\_Type \* *base*, sim\_clock\_gate\_name\_t *name* ) [inline], [static]

This function will get the clock gate state for specific module.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

## SIM HAL driver

|             |                            |
|-------------|----------------------------|
| <i>name</i> | Name of the module to get. |
|-------------|----------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**53.2.7.4 static void CLOCK\_HAL\_SetExternalRefClock32kSrc ( SIM\_Type \* *base*, clock\_er32k\_src\_t *setting* ) [inline], [static]**

This function sets the clock selection of ERCLK32K.

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>setting</i> | The value to set.                      |

**53.2.7.5 static clock\_er32k\_src\_t CLOCK\_HAL\_GetExternalRefClock32kSrc ( SIM\_Type \* *base* ) [inline], [static]**

This function gets the clock selection of ERCLK32K.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

Current selection.

**53.2.7.6 static void CLOCK\_HAL\_SetOsc32kOutSel ( SIM\_Type \* *base*, clock\_osc32kout\_sel\_t *setting* ) [inline], [static]**

This function sets ERCLK32K output pin.

Parameters

---

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>setting</i> | The value to set.                      |

**53.2.7.7 static clock\_osc32kout\_sel\_t CLOCK\_HAL\_GetOsc32kOutSel ( SIM\_Type \* *base* ) [inline], [static]**

This function gets ERCLK32K output pin setting.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

Current selection.

**53.2.7.8 static uint32\_t SIM\_HAL\_GetRamSize ( SIM\_Type \* *base* ) [inline], [static]**

This function gets the RAM size. The field specifies the amount of system RAM available on the device.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

size RAM size on the device

**53.2.7.9 static void CLOCK\_HAL\_SetPliflSel ( SIM\_Type \* *base*, clock\_pllfl\_sel\_t *setting* ) [inline], [static]**

This function sets the selection of the high frequency clock for various peripheral clocking options

Parameters

## SIM HAL driver

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>setting</i> | The value to set.                      |

### 53.2.7.10 **static clock\_pllfl\_sel\_t CLOCK\_HAL\_GetPllflSel ( SIM\_Type \* *base* ) [inline], [static]**

This function gets the selection of the high frequency clock for various peripheral clocking options

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

Current selection.

### 53.2.7.11 **static void CLOCK\_HAL\_SetTraceClkSrc ( SIM\_Type \* *base*, clock\_trace\_src\_t *setting* ) [inline], [static]**

This function sets debug trace clock selection.

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>setting</i> | The value to set.                      |

### 53.2.7.12 **static clock\_trace\_src\_t CLOCK\_HAL\_GetTraceClkSrc ( SIM\_Type \* *base* ) [inline], [static]**

This function gets debug trace clock selection.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

Current selection.

**53.2.7.13** `static void CLOCK_HAL_SetClkOutSel ( SIM_Type * base, 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>base</i>    | Base address for current SIM instance. |
| <i>setting</i> | The value to set.                      |

### 53.2.7.14 **static clock\_clkout\_src\_t CLOCK\_HAL\_GetClkOutSel ( SIM\_Type \* *base* ) [inline], [static]**

This function gets the selection of the clock to output on the CLKOUT pin.

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

### Returns

Current selection.

### 53.2.7.15 **static void CLOCK\_HAL\_SetOutDiv1 ( SIM\_Type \* *base*, uint8\_t *setting* ) [inline], [static]**

This function sets divide value OUTDIV1.

### Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>setting</i> | The value to set.                      |

### 53.2.7.16 **static uint8\_t CLOCK\_HAL\_GetOutDiv1 ( SIM\_Type \* *base* ) [inline], [static]**

This function gets divide value OUTDIV1.

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

### Returns

Current divide value.

```
53.2.7.17 static void CLOCK_HAL_SetOutDiv2 (SIM_Type * base, uint8_t setting)
 [inline], [static]
```

This function sets divide value OUTDIV2.

## SIM HAL driver

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>setting</i> | The value to set.                      |

**53.2.7.18** `static uint8_t CLOCK_HAL_GetOutDiv2 ( SIM_Type * base ) [inline], [static]`

This function gets divide value OUTDIV2.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

Current divide value.

**53.2.7.19** `static void CLOCK_HAL_SetOutDiv4 ( SIM_Type * base, uint8_t setting ) [inline], [static]`

This function sets divide value OUTDIV4.

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>setting</i> | The value to set.                      |

**53.2.7.20** `static uint8_t CLOCK_HAL_GetOutDiv4 ( SIM_Type * base ) [inline], [static]`

This function gets divide value OUTDIV4.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

Current divide value.



**53.2.7.21** void CLOCK\_HAL\_SetOutDiv ( SIM\_Type \* *base*, 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>base</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                    |

This function sets the setting for all clock out dividers at the same time. See the reference manual for a supported clock divider and value range and the `clock_divider_names_t` for clock out dividers.

### Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</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                    |

### 53.2.7.22 void CLOCK\_HAL\_GetOutDiv ( SIM\_Type \* *base*, 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>base</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                    |

### 53.2.7.23 void SIM\_HAL\_SetAdcAlternativeTriggerCmd ( SIM\_Type \* *base*, uint32\_t *instance*, bool *enable* ) [inline]

Sets the USB voltage regulator enabled setting.

This function enables/disables the alternative conversion triggers for ADCx.

## Parameters

|                 |                                                                                                                                                                                        |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |

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>base</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> |

Sets the ADCx alternate trigger enable setting.

This function enables/disables the alternative conversion triggers for ADCx.

## Parameters

|                 |                                                                                                                                                                                        |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |

### 53.2.7.24 **bool SIM\_HAL\_GetAdcAlternativeTriggerCmd ( SIM\_Type \* *base*, uint32\_t *instance* ) [inline]**

This function gets the ADCx alternate trigger enable setting.

Parameters

---

## SIM HAL driver

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

Returns

enabled True if ADCx alternate trigger is enabled

**53.2.7.25 void SIM\_HAL\_SetAdcPreTriggerMode ( SIM\_Type \* *base*, 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>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>select</i>   | pre-trigger select setting for ADCx    |

**53.2.7.26 sim\_adc\_pretrg\_sel\_t SIM\_HAL\_GetAdcPreTriggerMode ( SIM\_Type \* *base*, uint32\_t *instance* ) [inline]**

This function gets the ADCx pre-trigger select setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

Returns

select ADCx pre-trigger select setting

**53.2.7.27 void SIM\_HAL\_SetAdcTriggerMode ( SIM\_Type \* *base*, 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>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>select</i>   | trigger select setting for ADCx        |

**53.2.7.28 `sim_adc_trg_sel_t SIM_HAL_GetAdcTriggerMode ( SIM_Type * base, uint32_t instance ) [inline]`**

This function gets the ADCx trigger select setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

Returns

ADCx trigger select setting

**53.2.7.29 `void SIM_HAL_SetAdcTriggerModeOneStep ( SIM_Type * base, 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>base</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.                          |

**53.2.7.30 `void SIM_HAL_SetUartRxSrcMode ( SIM_Type * base, uint32_t instance, sim_uart_rxsrc_t select ) [inline]`**

This function selects the source for the UARTx receive data.

## SIM HAL driver

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>select</i>   | the source for the UARTx receive data  |

### 53.2.7.31 `sim_uart_rxsrc_t SIM_HAL_GetUartRxSrcMode ( SIM_Type * base, uint32_t instance ) [inline]`

This function gets the UARTx receive data source select setting.

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

### Returns

select UARTx receive data source select setting

### 53.2.7.32 `void SIM_HAL_SetUartTxSrcMode ( SIM_Type * base, uint32_t instance, sim_uart_txsrc_t select ) [inline]`

This function selects the source for the UARTx transmit data.

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>select</i>   | the source for the UARTx transmit data |

### 53.2.7.33 `sim_uart_txsrc_t SIM_HAL_GetUartTxSrcMode ( SIM_Type * base, uint32_t instance ) [inline]`

This function gets the UARTx transmit data source select setting.

## Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

## Returns

select UARTx transmit data source select setting

**53.2.7.34 void SIM\_HAL\_SetFtmTriggerSrcMode ( SIM\_Type \* *base*, uint32\_t *instance*, uint8\_t *trigger*, sim\_ftm\_trg\_src\_t *select* )**

This function selects the source of FTMx hardware trigger y.

## Parameters

|                 |                                                                                                                                                                       |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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"> <li>• 0: Pre-trigger A selected for ADCx.</li> <li>• 1: Pre-trigger B selected for ADCx.</li> </ul> |

**53.2.7.35 sim\_ftm\_trg\_src\_t SIM\_HAL\_GetFtmTriggerSrcMode ( SIM\_Type \* *base*, uint32\_t *instance*, uint8\_t *trigger* )**

This function gets the FlexTimer x hardware trigger y source select setting.

## Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</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

---

**SIM HAL driver**

**53.2.7.36** void **SIM\_HAL\_SetFtmExternalClkPinMode** ( **SIM\_Type** \* *base*, **uint32\_t** *instance*, **sim\_ftm\_clk\_sel\_t** *select* )

This function selects the source of FTMx external clock pin select.



## Parameters

|                 |                                                                                                                                                                                                 |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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"> <li>• 0: FTMx external clock driven by FTM CLKIN0 pin.</li> <li>• 1: FTMx external clock driven by FTM CLKIN1 pin.</li> </ul> |

**53.2.7.37 `sim_ftm_clk_sel_t` `SIM_HAL_GetFtmExternalClkPinMode ( SIM_Type * base, uint32_t instance )`**

This function gets the FlexTimer x external clock pin select setting.

## Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

## Returns

select FlexTimer x external clock pin select setting

**53.2.7.38 `void` `SIM_HAL_SetFtmChSrcMode ( SIM_Type * base, 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>base</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 |

**53.2.7.39 `sim_ftm_ch_src_t` `SIM_HAL_GetFtmChSrcMode ( SIM_Type * base, uint32_t instance, uint8_t channel )`**

This function gets the FlexTimer x channel y input capture source select setting.

## SIM HAL driver

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</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

### 53.2.7.40 void SIM\_HAL\_SetFtmFaultSelMode ( SIM\_Type \* *base*, uint32\_t *instance*, uint8\_t *fault*, sim\_ftm\_ft\_sel\_t *select* )

This function sets the FlexTimer x fault y select setting.

### Parameters

|                 |                                                                                                                                                                     |
|-----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |

### 53.2.7.41 sim\_ftm\_ft\_sel\_t SIM\_HAL\_GetFtmFaultSelMode ( SIM\_Type \* *base*, uint32\_t *instance*, uint8\_t *fault* )

This function gets the FlexTimer x fault y select setting.

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</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

**53.2.7.42** void SIM\_HAL\_SetFtmChOutSrcMode ( SIM\_Type \* *base*, uint32\_t *instance*,  
uint8\_t *channel*, sim\_ftm\_ch\_out\_src\_t *select* )

This function selects the FlexTimer x channel y output source.

## SIM HAL driver

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</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    |

### 53.2.7.43 `sim_ftm_ch_out_src_t SIM_HAL_GetFtmChOutSrcMode ( SIM_Type * base, uint32_t instance, uint8_t channel )`

This function gets the FlexTimer x channel y output source select setting.

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</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

### 53.2.7.44 `void SIM_HAL_SetFtmSyncCmd ( SIM_Type * base, uint32_t instance, bool sync )`

This function sets FlexTimer x hardware trigger 0 software synchronization. FTMxSYNCBIT.

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>sync</i>     | Synchronize or not.                    |

### 53.2.7.45 `static bool SIM_HAL_GetFtmSyncCmd ( SIM_Type * base, uint32_t instance ) [inline], [static]`

This function gets FlexTimer x hardware trigger 0 software synchronization. FTMxSYNCBIT.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

**53.2.7.46** `static uint32_t SIM_HAL_GetFamilyId ( SIM_Type * base ) [inline], [static]`

This function gets the Kinetis Family ID in the System Device ID register.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

id Kinetis Family ID

**53.2.7.47** `static uint32_t SIM_HAL_GetSubFamilyId ( SIM_Type * base ) [inline], [static]`

This function gets the Kinetis Sub-Family ID in System Device ID register.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

id Kinetis Sub-Family ID

**53.2.7.48** `static uint32_t SIM_HAL_GetSeriesId ( SIM_Type * base ) [inline], [static]`

This function gets the Kinetis Series ID in System Device ID register.

Parameters

---

## SIM HAL driver

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

id Kinetis Series ID

### 53.2.7.49 `static uint32_t SIM_HAL_GetRevId ( SIM_Type * base ) [inline], [static]`

This function gets the Kinetis Revision ID in System Device ID register.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

id Kinetis Revision ID

### 53.2.7.50 `static uint32_t SIM_HAL_GetDieId ( SIM_Type * base ) [inline], [static]`

This function gets the Kinetis Die ID in System Device ID register.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

id Kinetis Die ID

### 53.2.7.51 `static uint32_t SIM_HAL_GetFamId ( SIM_Type * base ) [inline], [static]`

This function gets the Kinetis family identification in System Device ID register.

Parameters

---

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

id Kinetis family identification

**53.2.7.52 static uint32\_t SIM\_HAL\_GetPinCntId ( SIM\_Type \* *base* ) [inline], [static]**

This function gets the Kinetis Pincount ID in System Device ID register.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

id Kinetis Pincount ID

**53.2.7.53 static uint32\_t SIM\_HAL\_GetProgramFlashSize ( SIM\_Type \* *base* ) [inline], [static]**

This function gets the program flash size in the Flash Configuration Register 1.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

size Program flash Size

**53.2.7.54 static void SIM\_HAL\_SetFlashDoze ( SIM\_Type \* *base*, uint32\_t *setting* ) [inline], [static]**

This function sets the Flash Doze in the Flash Configuration Register 1.

## SIM HAL driver

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>setting</i> | Flash Doze setting                     |

**53.2.7.55** `static uint32_t SIM_HAL_GetFlashDoze ( SIM_Type * base ) [inline], [static]`

This function gets the Flash Doze in the Flash Configuration Register 1.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

setting Flash Doze setting

**53.2.7.56** `static void SIM_HAL_SetFlashDisableCmd ( SIM_Type * base, bool disable ) [inline], [static]`

This function sets the Flash disable setting in the Flash Configuration Register 1.

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>disable</i> | Flash disable setting                  |

**53.2.7.57** `static bool SIM_HAL_GetFlashDisableCmd ( SIM_Type * base ) [inline], [static]`

This function gets the Flash disable setting in the Flash Configuration Register 1.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

setting Flash disable setting



```
53.2.7.58 static uint32_t SIM_HAL_GetFlashMaxAddrBlock0 (SIM_Type * base)
 [inline], [static]
```

This function gets the Flash maximum block 0 in Flash Configuration Register 2.

## SIM HAL driver

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

### Returns

address Flash maximum block 0 address

**53.2.7.59** `static void CLOCK_HAL_SetSdhcSrc ( SIM_Type * base, uint32_t instance, clock_sdhc_src_t setting ) [inline], [static]`

This function sets the SDHC clock source selection.

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |
| <i>setting</i>  | The value to set.                      |

**53.2.7.60** `static clock_sdhc_src_t CLOCK_HAL_GetSdhcSrc ( SIM_Type * base, uint32_t instance ) [inline], [static]`

This function gets the SDHC clock source selection.

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

### Returns

Current selection.

**53.2.7.61** `static void CLOCK_HAL_SetTimeSrc ( SIM_Type * base, uint32_t instance, clock_time_src_t setting ) [inline], [static]`

This function sets the ethernet timestamp clock source selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |
| <i>setting</i>  | The value to set.                      |

**53.2.7.62 static clock\_time\_src\_t CLOCK\_HAL\_GetTimeSrc ( SIM\_Type \* *base*, uint32\_t *instance* ) [inline], [static]**

This function gets the ethernet timestamp clock source selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

Returns

Current selection.

**53.2.7.63 static void CLOCK\_HAL\_SetRmiiSrc ( SIM\_Type \* *base*, uint32\_t *instance*, clock\_rmii\_src\_t *setting* ) [inline], [static]**

This function sets the Ethernet RMII interface clock source selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |
| <i>setting</i>  | The value to set.                      |

**53.2.7.64 static clock\_rmii\_src\_t CLOCK\_HAL\_GetRmiiSrc ( SIM\_Type \* *base*, uint32\_t *instance* ) [inline], [static]**

This function gets the Ethernet RMII interface clock source selection.

## SIM HAL driver

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

### Returns

Current selection.

**53.2.7.65** `static void CLOCK_HAL_SetRtcClkOutSel ( SIM_Type * base,  
clock_rtcout_src_t setting ) [inline], [static]`

This function sets the selection of the clock to output on the RTC\_CLKOUT pin.

### Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>setting</i> | The value to set.                      |

**53.2.7.66** `static clock_rtcout_src_t CLOCK_HAL_GetRtcClkOutSel ( SIM_Type * base )  
[inline], [static]`

This function gets the selection of the clock to output on the RTC\_CLKOUT pin.

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

### Returns

Current selection.

**53.2.7.67** `static void CLOCK_HAL_SetOutDiv3 ( SIM_Type * base, uint8_t setting )  
[inline], [static]`

This function sets divide value OUTDIV3.

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>setting</i> | The value to set.                      |

**53.2.7.68** `static uint8_t CLOCK_HAL_GetOutDiv3 ( SIM_Type * base ) [inline], [static]`

This function gets divide value OUTDIV3.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

Current divide value.

**53.2.7.69** `static void SIM_HAL_SetPtd7PadDriveStrengthMode ( SIM_Type * base, sim_ptd7pad_strengh_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>base</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> |

**53.2.7.70** `static sim_ptd7pad_strengh_t SIM_HAL_GetPtd7PadDriveStrengthMode ( SIM_Type * base ) [inline], [static]`

This function gets the PTD7 pad drive strength setting.

## SIM HAL driver

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

### Returns

setting PTD7 pad drive strength setting

#### 53.2.7.71 `static void SIM_HAL_SetFlexbusSecurityLevelMode ( SIM_Type * base, 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>base</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> |

#### 53.2.7.72 `static sim_flexbus_security_level_t SIM_HAL_GetFlexbusSecurityLevelMode ( SIM_Type * base ) [inline], [static]`

This function gets the FlexBus security level setting.

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

### Returns

setting FlexBus security level setting

#### 53.2.7.73 `static uint32_t SIM_HAL_GetFlexnvmSize ( SIM_Type * base ) [inline], [static]`

This function gets the FlexNVM size in the Flash Configuration Register 1.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

size FlexNVM Size

**53.2.7.74** `static uint32_t SIM_HAL_GetEepromSize ( SIM_Type * base ) [inline], [static]`

This function gets the EEPROM size in the Flash Configuration Register 1.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

size EEPROM Size

**53.2.7.75** `static uint32_t SIM_HAL_GetFlexnvmPartition ( SIM_Type * base ) [inline], [static]`

This function gets the FlexNVM partition in the Flash Configuration Register1

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

setting FlexNVM partition setting

**53.2.7.76** `static uint32_t SIM_HAL_GetFlashMaxAddrBlock1 ( SIM_Type * base ) [inline], [static]`

This function gets the Flash maximum block 1 in Flash Configuration Register 1.

## SIM HAL driver

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

address Flash maximum block 0 address

**53.2.7.77 static uint32\_t SIM\_HAL\_GetProgramFlashCmd ( SIM\_Type \* *base* )  
[inline], [static]**

This function gets the program flash maximum block 0 in Flash Configuration Register 1.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

status program flash status

**53.2.7.78 static bool SIM\_HAL\_GetSwapProgramFlash ( SIM\_Type \* *base* ) [inline],  
[static]**

This function gets the Swap program flash flag in the Flash Configuration Register 2.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

status - Swap program flash flag(Active or Inactive)

**53.2.7.79 void CLOCK\_HAL\_SetUsbfsDiv ( SIM\_Type \* *base*, uint8\_t *usbdiv*, uint8\_t  
*usbfrac* )**

This function sets USB FS divider setting. Divider output clock = Divider input clock \* [ (USBFSFRA-  
C+1) / (USBFSDIV+1) ]



Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>usbdiv</i>  | Value of USBFSDIV.                     |
| <i>usbfrac</i> | Value of USBFSFRAC.                    |

### 53.2.7.80 void CLOCK\_HAL\_GetUsbfsDiv ( SIM\_Type \* *base*, uint8\_t \* *usbdiv*, uint8\_t \* *usbfrac* )

This function gets USB FS divider setting. Divider output clock = Divider input clock \* [ (USBFSFRAC+1) / (USBFSDIV+1) ]

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>usbdiv</i>  | Value of USBFSDIV.                     |
| <i>usbfrac</i> | Value of USBFSFRAC.                    |

### 53.2.7.81 static void CLOCK\_HAL\_SetUsbfsSrc ( SIM\_Type \* *base*, 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>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |
| <i>setting</i>  | The value to set.                      |

### 53.2.7.82 static clock\_usbfs\_src\_t CLOCK\_HAL\_GetUsbfsSrc ( SIM\_Type \* *base*, uint32\_t *instance* ) [inline], [static]

This function gets the selection of the clock source for the USB FS 48 MHz clock.

Parameters

---

## SIM HAL driver

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

Returns

Current selection.

**53.2.7.83 static void SIM\_HAL\_SetUsbVoltRegulatorCmd ( SIM\_Type \* *base*, 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>base</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> |

**53.2.7.84 static bool SIM\_HAL\_GetUsbVoltRegulatorCmd ( SIM\_Type \* *base* ) [inline], [static]**

This function gets the USB voltage regulator enabled setting.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

enabled True if the USB voltage regulator is enabled.

**53.2.7.85 static void SIM\_HAL\_SetUsbVoltRegulatorInStdbyDuringStopMode ( SIM\_Type \* *base*, 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>base</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> |

**53.2.7.86** `static sim_usbsstby_mode_t SIM_HAL_GetUsbVoltRegulatorInStdbyDuringStopMode ( SIM_Type * base ) [inline], [static]`

This function gets the USB voltage regulator in a standby mode setting.

## Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

## Returns

setting USB voltage regulator in a standby mode setting

**53.2.7.87** `static void SIM_HAL_SetUsbVoltRegulatorInStdbyDuringVlprwMode ( SIM_Type * base, 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>base</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> |

**53.2.7.88** `static sim_usbvstby_mode_t SIM_HAL_GetUsbVoltRegulatorInStdbyDuringVlprwMode ( SIM_Type * base ) [inline], [static]`

This function gets the USB voltage regulator in a standby mode during the VLPR or the VLPW.

## SIM HAL driver

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

### Returns

setting USB voltage regulator in a standby mode during the VLPR or the VLPW

#### 53.2.7.89 **static void SIM\_HAL\_SetUsbVoltRegulatorInStdbyDuringStopCmd ( SIM\_Type \* *base*, 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>base</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> |

#### 53.2.7.90 **static bool SIM\_HAL\_GetUsbVoltRegulatorInStdbyDuringStopCmd ( SIM\_Type \* *base* ) [inline], [static]**

This function gets the USB voltage regulator stop standby write enable setting.

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

### Returns

enabled True if the USB voltage regulator stop standby write is enabled.

#### 53.2.7.91 **static void SIM\_HAL\_SetUsbVoltRegulatorInStdbyDuringVlprwCmd ( SIM\_Type \* *base*, 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>base</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> |

**53.2.7.92 static bool SIM\_HAL\_GetUsbVoltRegulatorInStdbyDuringVlprwCmd ( SIM\_Type \* *base* ) [inline], [static]**

This function gets the USB voltage regulator VLP standby write enable setting.

## Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

## Returns

enabled True if the USB voltage regulator VLP standby write is enabled.

**53.2.7.93 static void SIM\_HAL\_SetUsbVoltRegulatorWriteCmd ( SIM\_Type \* *base*, 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 SOPT1[USBREGEN].

## Parameters

|               |                                                                                                                                                                                                  |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |

**53.2.7.94 static bool SIM\_HAL\_GetUsbVoltRegulatorWriteCmd ( SIM\_Type \* *base* ) [inline], [static]**

This function gets the USB voltage regulator enable write enable setting.

## SIM HAL driver

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

### Returns

enabled True if USB voltage regulator enable write is enabled.

**53.2.7.95** `static void CLOCK_HAL_SetLpuartSrc ( SIM_Type * base, uint32_t instance, clock_lpuart_src_t setting ) [inline], [static]`

Set the clock selection of LPUART.

This function sets lpuart clock source selection.

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | LPUART instance.                       |
| <i>setting</i>  | The value to set.                      |

This function sets the clock selection of LPUART.

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | LPUART instance.                       |
| <i>setting</i>  | The value to set.                      |

**53.2.7.96** `static clock_lpuart_src_t CLOCK_HAL_GetLpuartSrc ( SIM_Type * base, uint32_t instance ) [inline], [static]`

Get the clock selection of LPUART.

This function gets lpuart clock source selection.

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

|                 |                  |
|-----------------|------------------|
| <i>instance</i> | LPUART instance. |
|-----------------|------------------|

Returns

Current selection.

This function gets the clock selection of LPUART.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | LPUART instance.                       |

Returns

Current selection.

**53.2.7.97 static void SIM\_HAL\_SetLpuartRxSrcMode ( SIM\_Type \* *base*, uint32\_t *instance*, sim\_lpuart\_rxsrc\_t *select* ) [inline], [static]**

This function selects the source for the LPUARTx receive data.

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | Base address for current SIM instance.  |
| <i>instance</i> | device instance.                        |
| <i>select</i>   | the source for the LPUARTx receive data |

This function selects the source for the LPUARTx receive data.

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | Register base address of SIM.           |
| <i>instance</i> | LPUART instance.                        |
| <i>select</i>   | the source for the LPUARTx receive data |

**53.2.7.98 static sim\_lpuart\_rxsrc\_t SIM\_HAL\_GetLpuartRxSrcMode ( SIM\_Type \* *base*, uint32\_t *instance* ) [inline], [static]**

This function gets the LPUARTx receive data source select setting.

## SIM HAL driver

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

### Returns

select LPUARTx receive data source select setting

This function gets the LPUARTx receive data source select setting.

### Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>base</i>     | Register base address of SIM. |
| <i>instance</i> | LPUART instance.              |

### Returns

select UARTx receive data source select setting

**53.2.7.99** `static void CLOCK_HAL_SetUsbhsSlowClockSrc ( SIM_Type * base, uint32_t instance, clock_usbhs_slowclk_src_t setting ) [inline], [static]`

This function sets the selection of the clock source for the USB HS/USB PHY slow clock.

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |
| <i>setting</i>  | The value to set.                      |

**53.2.7.100** `static clock_usbhs_slowclk_src_t CLOCK_HAL_GetUsbhsSlowClockSrc ( SIM_Type * base, uint32_t instance ) [inline], [static]`

This function gets the selection of the clock source for the USB HS/USB PHY slow clock.

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |



Returns

Current selection.

**53.2.7.101 void CLOCK\_HAL\_SetPIIFIDiv ( SIM\_Type \* *base*, uint8\_t *pllflldiv*, uint8\_t *pllflfrac* )**

This function sets PLL/FLL divider setting. Divider output clock = Divider input clock \* [ (PLLFLLFRAC+1) / (PLLFLLDIV+1) ]

Parameters

|                  |                                        |
|------------------|----------------------------------------|
| <i>base</i>      | Base address for current SIM instance. |
| <i>pllflldiv</i> | Value of PLLFLLDIV.                    |
| <i>pllflfrac</i> | Value of PLLFLLFRAC.                   |

**53.2.7.102 void CLOCK\_HAL\_GetPIIFIDiv ( SIM\_Type \* *base*, uint8\_t \* *pllflldiv*, uint8\_t \* *pllflfrac* )**

This function gets PLL/FLL divider setting. Divider output clock = Divider input clock \* [ (PLLFLLFRAC+1) / (PLLFLLDIV+1) ]

Parameters

|                  |                                        |
|------------------|----------------------------------------|
| <i>base</i>      | Base address for current SIM instance. |
| <i>pllflldiv</i> | Value of PLLFLLDIV.                    |
| <i>pllflfrac</i> | Value of PLLFLLFRAC.                   |

**53.2.7.103 void CLOCK\_HAL\_SetTraceDiv ( SIM\_Type \* *base*, uint8\_t *tracediv*, uint8\_t *tracefrac* )**

This function sets TRACECLK divider setting. Divider output clock = Divider input clock \* [ (TRACEFRAC+1) / (TRACEDIV+1) ]

Parameters

|                  |                                        |
|------------------|----------------------------------------|
| <i>base</i>      | Base address for current SIM instance. |
| <i>tracediv</i>  | Value of TRACEDIV.                     |
| <i>tracefrac</i> | Value of PLLFLLFRAC.                   |

**SIM HAL driver**

**53.2.7.104** void **CLOCK\_HAL\_GetTraceDiv** ( SIM\_Type \* *base*, uint8\_t \* *tracediv*, uint8\_t \* *tracefrac* )

This function gets TRACECLK divider setting. Divider output clock = Divider input clock \* [ (TRACE-FRAC+1) / (TRACEDIV+1) ]

## Parameters

|                  |                                        |
|------------------|----------------------------------------|
| <i>base</i>      | Base address for current SIM instance. |
| <i>tracediv</i>  | Value of PLLFLLDIV.                    |
| <i>tracefrac</i> | Value of PLLFLLFRAC.                   |

**53.2.7.105** `static void CLOCK_HAL_SetTpmSrc ( SIM_Type * base, uint32_t instance, clock_tpm_src_t setting ) [inline], [static]`

This function sets the TPM clock source selection.

## Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |
| <i>setting</i>  | The value to set.                      |

**53.2.7.106** `static clock_tpm_src_t CLOCK_HAL_GetTpmSrc ( SIM_Type * base, uint32_t instance ) [inline], [static]`

This function gets the TPM clock source selection.

## Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

## Returns

Current selection.

**53.2.7.107** `static void SIM_HAL_SetUsbVoltRegulatorInrushLimitCmd ( SIM_Type * base, bool enable ) [inline], [static]`

This function controls whether the USB voltage regulator inrush current limit is enabled.

## SIM HAL driver

### Parameters

|               |                                                                                                                                                                                                                                              |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | Base address for current SIM instance.                                                                                                                                                                                                       |
| <i>enable</i> | USB voltage regulator inrush limit enable setting <ul style="list-style-type: none"> <li>• true: USB voltage regulator inrush current limit is enabled.</li> <li>• false: USB voltage regulator inrush current limit is disabled.</li> </ul> |

### 53.2.7.108 static bool SIM\_HAL\_GetUsbVoltRegulatorInrushLimitCmd ( SIM\_Type \* *base* ) [inline], [static]

This function gets the USB voltage regulator inrush current limit enabled setting.

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

### Returns

enabled True if the USB voltage regulator is enabled.

### 53.2.7.109 static void SIM\_HAL\_SetUsbVoltRegulatorOutputTargetCmd ( SIM\_Type \* *base*, sim\_usbvout\_mode\_t *target* ) [inline], [static]

This function controls the USB voltage regulator output voltage.

### Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | Base address for current SIM instance. |
| <i>target</i> | USB voltage regulator output target    |

### 53.2.7.110 static sim\_usbvout\_mode\_t SIM\_HAL\_GetUsbVoltRegulatorOutputTargetCmd ( SIM\_Type \* *base* ) [inline], [static]

This function gets the USB voltage regulator output voltage.

## Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | Base address for current SIM instance. |
| <i>enable</i> | USB voltage regulator output target    |

**53.2.7.111 static void SIM\_HAL\_SetUsbPhyPIIRegulatorCmd ( SIM\_Type \* *base*, bool *enable* ) [inline], [static]**

This function controls whether the PLL regulator in the USB PHY is enabled. The regulator must be enabled before enabling the PLL in the USB HS PHY.

## Parameters

|               |                                                                                                                                                                                       |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | Base address for current SIM instance.                                                                                                                                                |
| <i>enable</i> | USB PHY PLL regulator enable setting <ul style="list-style-type: none"> <li>• true: USB PHY PLL regulator is enabled.</li> <li>• false: USB PHY PLL regulator is disabled.</li> </ul> |

**53.2.7.112 static bool SIM\_HAL\_GetUsbPhyPIIRegulatorCmd ( SIM\_Type \* *base* ) [inline], [static]**

This function gets the USB PHY PLL regulator enabled setting.

## Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

## Returns

enabled True if the USB PHY PLL regulator is enabled.

**53.2.7.113 static void SIM\_HAL\_SetLpuartTxSrcMode ( SIM\_Type \* *base*, uint32\_t *instance*, sim\_lpuart\_txsrc\_t *select* ) [inline], [static]**

This function selects the source for the LPUARTx transmit data.

## SIM HAL driver

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | Base address for current SIM instance.  |
| <i>instance</i> | device instance.                        |
| <i>select</i>   | the source for the LPUARTx receive data |

This function selects the source for the LPUARTx transmit data.

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | Register base address of SIM.           |
| <i>instance</i> | LPUART instance.                        |
| <i>select</i>   | the source for the UARTx transmit data. |

**53.2.7.114** `static sim_lpuart_rxsrc_t SIM_HAL_GetLpuartTxSrcMode ( SIM_Type * base, uint32_t instance ) [inline], [static]`

This function gets the LPUARTx transmit data source select setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

Returns

select LPUARTx transmit data source select setting

This function gets the LPUARTx transmit data source select setting.

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>base</i>     | Register base address of SIM. |
| <i>instance</i> | LPUART instance.              |

Returns

select UARTx transmit data source select setting.

**53.2.7.115** `static void CLOCK_HAL_SetLpisciSrc ( SIM_Type * base, uint32_t instance, clock_lpisci_src_t setting ) [inline], [static]`

This function sets the LPSCI clock source selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |
| <i>setting</i>  | The value to set.                      |

**53.2.7.116** `static clock_lpsci_src_t CLOCK_HAL_GetLpsciSrc ( SIM_Type * base, uint32_t instance ) [inline], [static]`

This function gets the LPSCI clock source selection.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

Returns

Current selection.

**53.2.7.117** `static void SIM_HAL_SetLpsciRxSrcMode ( SIM_Type * base, uint32_t instance, sim_lpsci_rxsrc_t select ) [inline], [static]`

This function selects the source for the LPSCIX receive data.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |
| <i>select</i>   | the source for the LPSCIX receive data |

**53.2.7.118** `static sim_lpsci_rxsrc_t SIM_HAL_GetLpsciRxSrcMode ( SIM_Type * base, uint32_t instance ) [inline], [static]`

This function gets the LPSCIX receive data source select setting.

## SIM HAL driver

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

### Returns

select LPSCIx receive data source select setting

**53.2.7.119** `static void SIM_HAL_SetLpSciTxSrcMode ( SIM_Type * base, uint32_t instance, sim_lpSciTxSrc_t select ) [inline], [static]`

This function selects the source for the LPSCIx transmit data.

### Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | Base address for current SIM instance.  |
| <i>instance</i> | device instance.                        |
| <i>select</i>   | the source for the LPSCIx transmit data |

**53.2.7.120** `static sim_lpSciTxSrc_t SIM_HAL_GetLpSciTxSrcMode ( SIM_Type * base, uint32_t instance ) [inline], [static]`

This function gets the LPSCIx transmit data source select setting.

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

### Returns

select LPSCIx transmit data source select setting

**53.2.7.121** `static uint32_t SIM_HAL_GetSramSize ( SIM_Type * base ) [inline], [static]`

This function gets the Kinetis SramSize in System Device ID register.



Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

id Kinetis SramSize

**53.2.7.122** `static void CLOCK_HAL_SetCopSrc ( SIM_Type * base, clock_cop_src_t setting ) [inline], [static]`

This function sets the clock selection of COP.

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>setting</i> | The value to set.                      |

**53.2.7.123** `static clock_cop_src_t CLOCK_HAL_GetCopSrc ( SIM_Type * base ) [inline], [static]`

This function gets the clock selection of COP.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

Current selection.

**53.2.7.124** `void SIM_HAL_SetLpuartOpenDrainCmd ( SIM_Type * base, uint32_t instance, bool enable ) [inline], [static]`

This function enables/disables the LPUARTx Open Drain.

Parameters

---

## SIM HAL driver

|                 |                                                                                                                                                                    |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |

### 53.2.7.125 `bool SIM_HAL_GetLpuartOpenDrainCmd ( SIM_Type * base, uint32_t instance ) [inline], [static]`

This function gets the LPUARTx Open Drain Enable setting.

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>base</i>     | Register base address of SIM. |
| <i>instance</i> | LPUART instance.              |

Returns

enabled True if LPUARTx Open Drain is enabled.

### 53.2.7.126 `void SIM_HAL_SetTpmChSrcMode ( SIM_Type * base, 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>base</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 |

### 53.2.7.127 `sim_tpm_ch_src_t SIM_HAL_GetTpmChSrcMode ( SIM_Type * base, uint32_t instance, uint8_t channel ) [inline], [static]`

This function gets the Timer/PWM x channel y input capture source select setting.

## Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</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

**53.2.7.128 void SIM\_HAL\_SetTpmExternalClkPinSelMode ( SIM\_Type \* *base*, 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>base</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> |

**53.2.7.129 sim\_tpm\_clk\_sel\_t SIM\_HAL\_GetTpmExternalClkPinSelMode ( SIM\_Type \* *base*, uint32\_t *instance* )**

This function gets the Timer/PWM x external clock pin select setting.

## Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

## Returns

select Timer/PWM x external clock pin select setting

**53.2.7.130 static void CLOCK\_HAL\_SetFlexioSrc ( SIM\_Type \* *base*, uint32\_t *instance*, clock\_flexio\_src\_t *setting* ) [inline], [static]**

This function selects the clock source for FLEXIO.

## SIM HAL driver

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |
| <i>setting</i>  | The value to set.                      |

### 53.2.7.131 `static clock_flexio_src_t CLOCK_HAL_GetFlexioSrc ( SIM_Type * base, uint32_t instance ) [inline], [static]`

This function gets the clock source of FLEXIO.

### Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | IP instance.                           |

### Returns

Current selection.

### 53.2.7.132 `static void SIM_HAL_SetUartOpenDrainCmd ( SIM_Type * base, uint32_t instance, bool enable ) [inline], [static]`

This function enables/disables the UARTx Open Drain.

### Parameters

|                 |                                                                                                                                                              |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>     | Register base address of SIM.                                                                                                                                |
| <i>instance</i> | UART instance.                                                                                                                                               |
| <i>enable</i>   | Enable/disable UARTx Open Drain <ul style="list-style-type: none"> <li>• True: Enable UARTx Open Drain</li> <li>• False: Disable UARTx Open Drain</li> </ul> |

### 53.2.7.133 `static bool SIM_HAL_GetUartOpenDrainCmd ( SIM_Type * base, uint32_t instance ) [inline], [static]`

This function gets the UARTx Open Drain Enable setting.

Parameters

|                 |                               |
|-----------------|-------------------------------|
| <i>base</i>     | Register base address of SIM. |
| <i>instance</i> | UART instance.                |

Returns

enabled True if UARTx Open Drain is enabled.

**53.2.7.134** `static uint32_t SIM_HAL_GetSramSizeId ( SIM_Type * base ) [inline], [static]`

This function gets the Kinetis SRAMSIZE ID in System Device ID register.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

id Kinetis SRAMSIZE ID

**53.2.7.135** `static void CLOCK_HAL_SetOutDiv5ENCmd ( SIM_Type * base, bool setting ) [inline], [static]`

This function sets divide value OUTDIV5EN.

Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>setting</i> | The value to set.                      |

**53.2.7.136** `static bool CLOCK_HAL_GetOutDiv5ENCmd ( SIM_Type * base ) [inline], [static]`

This function gets divide value OUTDIV5EN.

## SIM HAL driver

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

### Returns

Current divide value.

**53.2.7.137** `static void CLOCK_HAL_SetOutDiv5 ( SIM_Type * base, uint8_t setting ) [inline], [static]`

This function sets divide value OUTDIV5.

### Parameters

|                |                                        |
|----------------|----------------------------------------|
| <i>base</i>    | Base address for current SIM instance. |
| <i>setting</i> | The value to set.                      |

**53.2.7.138** `static uint8_t CLOCK_HAL_GetOutDiv5 ( SIM_Type * base ) [inline], [static]`

This function gets divide value OUTDIV5.

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

### Returns

Current divide value.

**53.2.7.139** `void CLOCK_HAL_SetAdcAltClkSrc ( SIM_Type * base, uint32_t instance, clock_adc_alt_src_t adcAltSrcSel )`

This function sets the ADC ALT clock source selection setting.

### Parameters

---

|                     |                                        |
|---------------------|----------------------------------------|
| <i>base</i>         | Base address for current SIM instance. |
| <i>instance</i>     | ADC module instance.                   |
| <i>adcAltSrcSel</i> | ADC ALT clock source.                  |

**53.2.7.140 clock\_adc\_alt\_src\_t CLOCK\_HAL\_GetAdcAltClkSrc ( SIM\_Type \* *base*, uint32\_t *instance* )**

This function gets the ADC ALT clock source selection setting.

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | ADC module instance.                   |

Returns

the ADC ALT clock source selection.

**53.2.7.141 void SIM\_HAL\_SetUartOpenDrainMode ( SIM\_Type \* *base*, uint32\_t *instance*, bool *enable* )**

This function enables/disables open drain for UARTx.

Parameters

|                 |                                                                                                                                                        |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |

**53.2.7.142 bool SIM\_HAL\_GetUartOpenDrainMode ( SIM\_Type \* *base*, uint32\_t *instance* )**

This function Gets the UARTx open drain enable setting for UARTx.

## SIM HAL driver

Parameters

|                 |                                        |
|-----------------|----------------------------------------|
| <i>base</i>     | Base address for current SIM instance. |
| <i>instance</i> | device instance.                       |

**53.2.7.143** `static void CLOCK_HAL_SetFtmFixFreqClkSrc ( SIM_Type * base,  
clock_ftm_fixedfreq_src_t ftmFixedFreqSel ) [inline], [static]`

This function sets the FTM Fixed clock source selection setting.

Parameters

|                        |                                        |
|------------------------|----------------------------------------|
| <i>base</i>            | Base address for current SIM instance. |
| <i>ftmFixedFreqSel</i> | FTM Fixed clock source.                |

**53.2.7.144** `static clock_ftm_fixedfreq_src_t CLOCK_HAL_GetFtmFixFreqClkSrc ( SIM_Type * base ) [inline], [static]`

This function gets the FTM Fixed clock source selection setting.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

the FTM Fixed clock source selection.

**53.2.7.145** `static void SIM_HAL_SetFtmCarrierFreqMode ( SIM_Type * base,  
sim_ftm_ftl_carrier_sel_t select ) [inline], [static]`

This function Sets the Carrier frequency selection for FTM0/2 output channel.

Parameters

|               |                                                                                                                                                                                                            |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</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> |



**53.2.7.146** `static sim_ftm_ft_carrier_sel_t SIM_HAL_GetFtmCarrierFreqMode ( SIM_Type * base ) [inline], [static]`

This function gets Carrier frequency selection setting for FTM0/2 output channel.

## SIM HAL driver

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

Carrier frequency selection;

**53.2.7.147** `static uint32_t SIM_HAL_GetSubFamId ( SIM_Type * base ) [inline], [static]`

This function gets the Kinetis SubFam ID in System Device ID register.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

id Kinetis SubFam ID

**53.2.7.148** `static uint32_t SIM_HAL_GetFlashMaxAddrBlock ( SIM_Type * base ) [inline], [static]`

This function gets the Flash maximum block in Flash Configuration Register 2.

Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

Returns

address Flash maximum block address

**53.2.7.149** `sim_hal_status_t CLOCK_HAL_SetSource ( SIM_Type * base, clock_source_names_t clockSource, uint8_t setting )`

This function sets the settings for a specified clock source. Each clock source has its own clock selection settings. See the chip reference manual for clock source detailed settings and the `clock_source_names_t` for clock sources.

## Parameters

|                    |                                                                    |
|--------------------|--------------------------------------------------------------------|
| <i>base</i>        | Base address for current SIM instance.                             |
| <i>clockSource</i> | Clock source name defined in <code>sim_clock_source_names_t</code> |
| <i>setting</i>     | Setting value                                                      |

## Returns

status If the clock source doesn't exist, it returns an error.

**53.2.7.150 `sim_hal_status_t` `CLOCK_HAL_GetSource` ( `SIM_Type` \* *base*, `clock_source_names_t` *clockSource*, `uint8_t` \* *setting* )**

This function gets the settings for a specified clock source. Each clock source has its own clock selection settings. See the reference manual for clock source detailed settings and the `clock_source_names_t` for clock sources.

## Parameters

|                    |                                        |
|--------------------|----------------------------------------|
| <i>base</i>        | Base address for current SIM instance. |
| <i>clockSource</i> | Clock source name                      |
| <i>setting</i>     | Current setting for the clock source   |

## Returns

status If the clock source doesn't exist, it returns an error.

**53.2.7.151 `sim_hal_status_t` `CLOCK_HAL_SetDivider` ( `SIM_Type` \* *base*, `clock_divider_names_t` *clockDivider*, `uint32_t` *setting* )**

This function sets the setting for a specified clock divider. See the reference manual for a supported clock divider and value range and the `clock_divider_names_t` for dividers.

## Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SIM instance. |
|-------------|----------------------------------------|

## SIM HAL driver

|                     |                    |
|---------------------|--------------------|
| <i>clockDivider</i> | Clock divider name |
| <i>setting</i>      | Divider setting    |

### Returns

status If the clock divider doesn't exist, it returns an error.

## 53.2.8 K02F12810 SIM HAL driver

### 53.2.8.1 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](#) {  
  [kClockPllFlSelIFll](#) = 0U,  
  [kClockPllFlSelIrc48M](#) = 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_rxsrc_k02f12810_t` {
  - kSimUartRxsrcPin,
  - kSimUartRxsrcCmp0,
  - kSimUartRxsrcCmp1 }

*SIM UART receive data source select.*
- enum `sim_uart_txsrc_k02f12810_t` {
  - kSimUartTxsrcPin,
  - kSimUartTxsrcFtm1,
  - kSimUartTxsrcFtm2 }

*SIM UART 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_ft_sel_k02f12810_t` {  
kSimFtmFtSel0,  
kSimFtmFtSel1 }
- 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.*

### 53.2.8.2 Macro Definition Documentation

53.2.8.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.8.3 Enumeration Type Documentation

#### 53.2.8.3.1 enum `clock_wdog_src_k02f12810_t`

Enumerator

- kClockWdogSrcLpoClk* LPO.
- kClockWdogSrcAltClk* Alternative clock, for this SOC it is Bus clock.

#### 53.2.8.3.2 enum `clock_trace_src_k02f12810_t`

Enumerator

- kClockTraceSrcMcgoutClk* MCG out clock.
- kClockTraceSrcCoreClk* core clock

## SIM HAL driver

### 53.2.8.3.3 enum clock\_port\_filter\_src\_k02f12810\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.  
*kClockPortFilterSrcLpoClk* LPO.

### 53.2.8.3.4 enum clock\_lptmr\_src\_k02f12810\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 53.2.8.3.5 enum clock\_pllfl\_sel\_k02f12810\_t

Enumerator

*kClockPlIFllSelFll* Fll clock.  
*kClockPlIFllSelIrc48M* IRC48MCLK.

### 53.2.8.3.6 enum clock\_er32k\_src\_k02f12810\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.8.3.7 enum clock\_clkout\_src\_k02f12810\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCGIRCLK.  
*kClockClkoutSelOsc0erClk* OSC0ERCLK.  
*kClockClkoutSelIrc48M* IRC48MCLK.



### 53.2.8.3.8 enum clock\_osc32kout\_sel\_k02f12810\_t

Enumerator

- kClockOsc32koutNone* ERCLK32K is not output.
- kClockOsc32koutPte0* ERCLK32K is output on PTE0.
- kClockOsc32koutPte26* ERCLK32K is output on PTE26.

### 53.2.8.3.9 enum sim\_adc\_pretrg\_sel\_k02f12810\_t

Enumerator

- kSimAdcPretrgselA* Pre-trigger A selected for ADCx.
- kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

### 53.2.8.3.10 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.

### 53.2.8.3.11 enum sim\_uart\_rxsrc\_k02f12810\_t

Enumerator

- kSimUartRxsrcPin* UARTx\_RX Pin.
- kSimUartRxsrcCmp0* CMP0.
- kSimUartRxsrcCmp1* CMP1.

### 53.2.8.3.12 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.

### 53.2.8.3.13 enum sim\_ftm\_trg\_src\_k02f12810\_t

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.

*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

### 53.2.8.3.14 enum sim\_ftm\_clk\_sel\_k02f12810\_t

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.

*kSimFtmClkSel1* FTM CLKIN1 pin.

### 53.2.8.3.15 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.

### 53.2.8.3.16 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.

### 53.2.8.3.17 enum sim\_ftmflt\_sel\_k02f12810\_t

Enumerator

*kSimFtmFltSel0* FlexTimer x fault y select 0.

*kSimFtmFltSel1* FlexTimer x fault y select 1.

**53.2.8.3.18 enum sim\_tpm\_clk\_sel\_k02f12810\_t**

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.**53.2.8.3.19 enum sim\_tpm\_ch\_src\_k02f12810\_t**

Enumerator

*kSimTpmChSrc0* TPMx\_CH0 signal.*kSimTpmChSrc1* CMP0 output.**53.2.8.3.20 enum sim\_clock\_gate\_name\_k02f12810\_t**

## SIM HAL driver

### 53.2.9 K22F12810 SIM HAL driver

#### 53.2.9.1 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_pllfl1_sel_k22f12810_t` {  
`kClockPllFl1SelFl1` = 0U,  
`kClockPllFl1SelIrc48M` = 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_rxsrc_k22f12810_t` {  
  - kSimLpuartRxsrcPin,
  - kSimLpuartRxsrcCmp0,
  - kSimLpuartRxsrcCmp1 }*SIM LPUART RX source.*
- enum `sim_uart_rxsrc_k22f12810_t` {  
  - kSimUartRxsrcPin,
  - kSimUartRxsrcCmp0,
  - kSimUartRxsrcCmp1 }*SIM UART receive data source select.*
- enum `sim_uart_txsrc_k22f12810_t` {  
  - kSimUartTxsrcPin,
  - kSimUartTxsrcFtm1,
  - kSimUartTxsrcFtm2 }*SIM UART 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_ft_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_streng_h_k22f12810_t` {  
`kSimCmtuartSinglePad`,  
`kSimCmtuartDualPad` }
- SIM CMT/UART pad drive strength.*

  - enum `sim_ptd7pad_streng_h_k22f12810_t` {  
`kSimPtd7padSinglePad`,  
`kSimPtd7padDualPad` }
- SIM PTD7 pad drive strength.*

  - enum `sim_flexbus_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.*

### 53.2.9.2 Macro Definition Documentation

53.2.9.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.9.3 Enumeration Type Documentation

53.2.9.3.1 `enum clock_wdog_src_k22f12810_t`

Enumerator

*kClockWdogSrcLpoClk* LPO.

*kClockWdogSrcAltClk* Alternative clock, for this SOC it is Bus clock.

## SIM HAL driver

### 53.2.9.3.2 enum clock\_trace\_src\_k22f12810\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock

### 53.2.9.3.3 enum clock\_port\_filter\_src\_k22f12810\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.  
*kClockPortFilterSrcLpoClk* LPO.

### 53.2.9.3.4 enum clock\_lptmr\_src\_k22f12810\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 53.2.9.3.5 enum clock\_usbfs\_src\_k22f12810\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)  
*kClockUsbfsSrcPllFllSel* Clock divider USB FS clock.

### 53.2.9.3.6 enum clock\_lpuart\_src\_k22f12810\_t

Enumerator

*kClockLpuartSrcNone* Clock disabled.  
*kClockLpuartSrcPllFllSel* Clock as selected by SOPT2[PLLFLSEL].  
*kClockLpuartSrcOsc0erClk* OSCERCLK.  
*kClockLpuartSrcMcgIrClk* MCGIRCLK.



### 53.2.9.3.7 enum clock\_sai\_src\_k22f12810\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllFlSel* MCGPLLCLK.

### 53.2.9.3.8 enum clock\_pllfl\_sel\_k22f12810\_t

Enumerator

*kClockPllFlSelFll* Fll clock.  
*kClockPllFlSelIrc48M* IRC48MCLK.

### 53.2.9.3.9 enum clock\_er32k\_src\_k22f12810\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.9.3.10 enum clock\_clkout\_src\_k22f12810\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCGIRCLK.  
*kClockClkoutSelRtc* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSC0ERCLK.  
*kClockClkoutSelIrc48M* IRC48MCLK.

### 53.2.9.3.11 enum clock\_rtcout\_src\_k22f12810\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

## SIM HAL driver

### 53.2.9.3.12 enum clock\_osc32kout\_sel\_k22f12810\_t

Enumerator

- kClockOsc32koutNone* ERCLK32K is not output.
- kClockOsc32koutPte0* ERCLK32K is output on PTE0.
- kClockOsc32koutPte26* ERCLK32K is output on PTE26.

### 53.2.9.3.13 enum sim\_usbsstby\_mode\_k22f12810\_t

Enumerator

- kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes
- kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

### 53.2.9.3.14 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

### 53.2.9.3.15 enum sim\_adc\_pretrg\_sel\_k22f12810\_t

Enumerator

- kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.
- kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.9.3.16 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.

**53.2.9.3.17 enum sim\_lpuart\_rxsrc\_k22f12810\_t**

Enumerator

*kSimLpuartRxsrcPin* LPUARTx\_RX Pin.  
*kSimLpuartRxsrcCmp0* CMP0.  
*kSimLpuartRxsrcCmp1* CMP1.

**53.2.9.3.18 enum sim\_uart\_rxsrc\_k22f12810\_t**

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.  
*kSimUartRxsrcCmp0* CMP0.  
*kSimUartRxsrcCmp1* CMP1.

**53.2.9.3.19 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.

**53.2.9.3.20 enum sim\_ftm\_trg\_src\_k22f12810\_t**

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.  
*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

**53.2.9.3.21 enum sim\_ftm\_clk\_sel\_k22f12810\_t**

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.  
*kSimFtmClkSel1* FTM CLKIN1 pin.

## SIM HAL driver

### 53.2.9.3.22 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.

### 53.2.9.3.23 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.

### 53.2.9.3.24 enum sim\_ftmflt\_sel\_k22f12810\_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

### 53.2.9.3.25 enum sim\_tpm\_clk\_sel\_k22f12810\_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 53.2.9.3.26 enum sim\_tpm\_ch\_src\_k22f12810\_t

Enumerator

- kSimTpmChSrc0* TPMx\_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

### 53.2.9.3.27 enum sim\_cmtuartpad\_streng\_h\_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.

**53.2.9.3.28 enum sim\_ptd7pad\_strength\_k22f12810\_t**

Enumerator

*kSimPtd7padSinglePad* Single-pad drive strength for PTD7.*kSimPtd7padDualPad* Dual-pad drive strength for PTD7.**53.2.9.3.29 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.**53.2.9.3.30 enum sim\_clock\_gate\_name\_k22f12810\_t**

## SIM HAL driver

### 53.2.10 K22F25612 SIM HAL driver

#### 53.2.10.1 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_pllfl1_sel_k22f25612_t` {  
`kClockPlIFllSelFll` = 0U,  
`kClockPlIFllSelPll` = 1U,  
`kClockPlIFllSelIrc48M` = 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` {  
`kSimUsbsstbyNoRegulator`,  
`kSimUsbsstbyWithRegulator` }  
*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_rxsrc_k22f25612_t` {  
  - kSimLpuartRxsrcPin,
  - kSimLpuartRxsrcCmp0,
  - kSimLpuartRxsrcCmp1 }*SIM LPUART RX source.*
- enum `sim_uart_rxsrc_k22f25612_t` {  
  - kSimUartRxsrcPin,
  - kSimUartRxsrcCmp0,
  - kSimUartRxsrcCmp1 }*SIM UART receive data source select.*
- enum `sim_uart_txsrc_k22f25612_t` {  
  - kSimUartTxsrcPin,
  - kSimUartTxsrcFtm1,
  - kSimUartTxsrcFtm2 }*SIM UART 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_ft_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_h_k22f25612_t` {  
`kSimCmtuartSinglePad`,  
`kSimCmtuartDualPad` }
- SIM CMT/UART pad drive strength.*

  - enum `sim_ptd7pad_streng_h_k22f25612_t` {  
`kSimPtd7padSinglePad`,  
`kSimPtd7padDualPad` }
- SIM PTD7 pad drive strength.*

  - enum `sim_flexbus_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.*

### 53.2.10.2 Macro Definition Documentation

53.2.10.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.10.3 Enumeration Type Documentation

53.2.10.3.1 enum `clock_wdog_src_k22f25612_t`

Enumerator

*kClockWdogSrcLpoClk* LPO.

*kClockWdogSrcAltClk* Alternative clock, for this SOC it is Bus clock.

## SIM HAL driver

### 53.2.10.3.2 enum clock\_trace\_src\_k22f25612\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock

### 53.2.10.3.3 enum clock\_port\_filter\_src\_k22f25612\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.  
*kClockPortFilterSrcLpoClk* LPO.

### 53.2.10.3.4 enum clock\_lptmr\_src\_k22f25612\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 53.2.10.3.5 enum clock\_usbfs\_src\_k22f25612\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)  
*kClockUsbfsSrcPllFllSel* Clock divider USB FS clock.

### 53.2.10.3.6 enum clock\_lpuart\_src\_k22f25612\_t

Enumerator

*kClockLpuartSrcNone* Clock disabled.  
*kClockLpuartSrcPllFllSel* Clock as selected by SOPT2[PLLFLSEL].  
*kClockLpuartSrcOsc0erClk* OSCERCLK.  
*kClockLpuartSrcMcgIrClk* MCGIRCLK.

### 53.2.10.3.7 enum clock\_sai\_src\_k22f25612\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllFllSel* MCGPLLCLK.

### 53.2.10.3.8 enum clock\_pllfl\_sel\_k22f25612\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.  
*kClockPllFllSelIrc48M* IRC48MCLK.

### 53.2.10.3.9 enum clock\_er32k\_src\_k22f25612\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.10.3.10 enum clock\_clkout\_src\_k22f25612\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCGIRCLK.  
*kClockClkoutSelRtc* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSC0ERCLK.  
*kClockClkoutSelIrc48M* IRC48MCLK.

### 53.2.10.3.11 enum clock\_rtcout\_src\_k22f25612\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

## SIM HAL driver

### 53.2.10.3.12 enum clock\_osc32kout\_sel\_k22f25612\_t

Enumerator

- kClockOsc32koutNone* ERCLK32K is not output.
- kClockOsc32koutPte0* ERCLK32K is output on PTE0.
- kClockOsc32koutPte26* ERCLK32K is output on PTE26.

### 53.2.10.3.13 enum sim\_usbsstby\_mode\_k22f25612\_t

Enumerator

- kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes
- kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

### 53.2.10.3.14 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

### 53.2.10.3.15 enum sim\_adc\_pretrg\_sel\_k22f25612\_t

Enumerator

- kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.
- kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.10.3.16 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.

### 53.2.10.3.17 enum sim\_lpuart\_rxsrc\_k22f25612\_t

Enumerator

*kSimLpuartRxsrcPin* LPUARTx\_RX Pin.  
*kSimLpuartRxsrcCmp0* CMP0.  
*kSimLpuartRxsrcCmp1* CMP1.

### 53.2.10.3.18 enum sim\_uart\_rxsrc\_k22f25612\_t

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.  
*kSimUartRxsrcCmp0* CMP0.  
*kSimUartRxsrcCmp1* CMP1.

### 53.2.10.3.19 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.

### 53.2.10.3.20 enum sim\_ftm\_trg\_src\_k22f25612\_t

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.  
*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

### 53.2.10.3.21 enum sim\_ftm\_clk\_sel\_k22f25612\_t

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.  
*kSimFtmClkSel1* FTM CLKIN1 pin.

## SIM HAL driver

### 53.2.10.3.22 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.

### 53.2.10.3.23 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.

### 53.2.10.3.24 enum sim\_ftmflt\_sel\_k22f25612\_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

### 53.2.10.3.25 enum sim\_tpm\_clk\_sel\_k22f25612\_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 53.2.10.3.26 enum sim\_tpm\_ch\_src\_k22f25612\_t

Enumerator

- kSimTpmChSrc0* TPMx\_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

### 53.2.10.3.27 enum sim\_cmtuartpad\_strength\_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.

**53.2.10.3.28 enum sim\_ptd7pad\_strength\_k22f25612\_t**

Enumerator

*kSimPtd7padSinglePad* Single-pad drive strength for PTD7.*kSimPtd7padDualPad* Dual-pad drive strength for PTD7.**53.2.10.3.29 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.**53.2.10.3.30 enum sim\_clock\_gate\_name\_k22f25612\_t**

## SIM HAL driver

### 53.2.11 K22F51212 SIM HAL driver

#### 53.2.11.1 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` {  
`kSimUsbsstbyNoRegulator`,  
`kSimUsbsstbyWithRegulator` }  
*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 UART receive data source select.*

- enum `sim_uart_txsrc_k22f51212_t` {  
`kSimUartTxsrcPin`,  
`kSimUartTxsrcFtm1`,  
`kSimUartTxsrcFtm2` }

*SIM UART 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_ft_sel_k22f51212_t` {
  - `kSimFtmFtmSel0`,
  - `kSimFtmFtmSel1` }
  - 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.*

### 53.2.11.2 Macro Definition Documentation

53.2.11.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.11.3 Enumeration Type Documentation

53.2.11.3.1 enum `clock_wdog_src_k22f51212_t`

Enumerator

*kClockWdogSrcLpoClk* LPO.

*kClockWdogSrcAltClk* Alternative clock, for this SOC it is Bus clock.

## SIM HAL driver

### 53.2.11.3.2 enum clock\_trace\_src\_k22f51212\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.  
*kClockTraceSrcCoreClk* core clock

### 53.2.11.3.3 enum clock\_port\_filter\_src\_k22f51212\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.  
*kClockPortFilterSrcLpoClk* LPO.

### 53.2.11.3.4 enum clock\_lptmr\_src\_k22f51212\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 53.2.11.3.5 enum clock\_usbfs\_src\_k22f51212\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)  
*kClockUsbfsSrcPllFllSel* Clock divider USB FS clock.

### 53.2.11.3.6 enum clock\_lpuart\_src\_k22f51212\_t

Enumerator

*kClockLpuartSrcNone* Clock disabled.  
*kClockLpuartSrcPllFllSel* Clock as selected by SOPT2[PLLFLSEL].  
*kClockLpuartSrcOsc0erClk* OSCERCLK.  
*kClockLpuartSrcMcgIrClk* MCGIRCLK.

### 53.2.11.3.7 enum clock\_sai\_src\_k22f51212\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllFllSel* MCGPLLFLCLK.

### 53.2.11.3.8 enum clock\_pllfl\_sel\_k22f51212\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.  
*kClockPllFllSelIrc48M* IRC48MCLK.

### 53.2.11.3.9 enum clock\_er32k\_src\_k22f51212\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.11.3.10 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.

### 53.2.11.3.11 enum clock\_rtcout\_src\_k22f51212\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

## SIM HAL driver

### 53.2.11.3.12 enum clock\_osc32kout\_sel\_k22f51212\_t

Enumerator

- kClockOsc32koutNone* ERCLK32K is not output.
- kClockOsc32koutPte0* ERCLK32K is output on PTE0.
- kClockOsc32koutPte26* ERCLK32K is output on PTE26.

### 53.2.11.3.13 enum sim\_usbsstby\_mode\_k22f51212\_t

Enumerator

- kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes
- kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

### 53.2.11.3.14 enum sim\_usbvstby\_mode\_k22f51212\_t

Enumerator

- kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes
- kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes

### 53.2.11.3.15 enum sim\_adc\_pretrg\_sel\_k22f51212\_t

Enumerator

- kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.
- kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.11.3.16 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.

### 53.2.11.3.17 enum sim\_lpuart\_rxsrc\_k22f51212\_t

Enumerator

*kSimLpuartRxsSrcPin* LPUARTx\_RX Pin.  
*kSimLpuartRxsSrcCmp0* CMP0.  
*kSimLpuartRxsSrcCmp1* CMP1.

### 53.2.11.3.18 enum sim\_uart\_rxsrc\_k22f51212\_t

Enumerator

*kSimUartRxsSrcPin* UARTx\_RX Pin.  
*kSimUartRxsSrcCmp0* CMP0.  
*kSimUartRxsSrcCmp1* CMP1.

### 53.2.11.3.19 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.

### 53.2.11.3.20 enum sim\_ftm\_trg\_src\_k22f51212\_t

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.  
*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

### 53.2.11.3.21 enum sim\_ftm\_clk\_sel\_k22f51212\_t

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.  
*kSimFtmClkSel1* FTM CLKIN1 pin.

## SIM HAL driver

### 53.2.11.3.22 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.

### 53.2.11.3.23 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.

### 53.2.11.3.24 enum sim\_ftmflt\_sel\_k22f51212\_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

### 53.2.11.3.25 enum sim\_tpm\_clk\_sel\_k22f51212\_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 53.2.11.3.26 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.

### 53.2.11.3.27 enum sim\_cmtuartpad\_strength\_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.



**53.2.11.3.28 enum sim\_ptd7pad\_strength\_k22f51212\_t**

Enumerator

*kSimPtd7padSinglePad* Single-pad drive strength for PTD7.*kSimPtd7padDualPad* Dual-pad drive strength for PTD7.**53.2.11.3.29 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.**53.2.11.3.30 enum sim\_clock\_gate\_name\_k22f51212\_t**

## SIM HAL driver

### 53.2.12 K10D10 SIM HAL driver

#### 53.2.12.1 Overview

The section describes the enumerations, macros and data structures for K10D10 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MK10D10.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_k10d10\\_t](#) {  
[kClockWdogSrcLpoClk](#),  
[kClockWdogSrcAltClk](#) }  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_k10d10\\_t](#) {  
[kClockTraceSrcMcgoutClk](#),  
[kClockTraceSrcCoreClk](#) }  
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_k10d10\\_t](#) {  
[kClockPortFilterSrcBusClk](#),  
[kClockPortFilterSrcLpoClk](#) }  
*PORTx digital input filter clock source select.*
- enum [clock\\_lptmr\\_src\\_k10d10\\_t](#) {  
[kClockLptmrSrcMcgIrClk](#),  
[kClockLptmrSrcLpoClk](#),  
[kClockLptmrSrcEr32kClk](#),  
[kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_time\\_src\\_k10d10\\_t](#) {  
[kClockTimeSrcCoreSysClk](#),  
[kClockTimeSrcPIIFilSel](#),  
[kClockTimeSrcOsc0erClk](#),  
[kClockTimeSrcExt](#) }  
*SIM timestamp clock source.*
- enum [clock\\_rmii\\_src\\_k10d10\\_t](#) {  
[kClockRmiiSrcExtalClk](#),  
[kClockRmiiSrcExt](#) }  
*SIM RMII clock source.*

- enum `clock_flexcan_src_k10d10_t` {  
`kClockFlexcanSrcOsc0erClk`,  
`kClockFlexcanSrcBusClk` }  
*FLEXCAN clock source select.*
- enum `clock_sdhc_src_k10d10_t` {  
`kClockSdhcSrcCoreSysClk`,  
`kClockSdhcSrcPllFllSel`,  
`kClockSdhcSrcOsc0erClk`,  
`kClockSdhcSrcExt` }  
*SDHC clock source.*
- enum `clock_sai_src_k10d10_t` {  
`kClockSaiSrcSysClk` = 0U,  
`kClockSaiSrcOsc0erClk` = 1U,  
`kClockSaiSrcPllClk` = 3U }  
*SAI clock source.*
- enum `clock_tsi_active_mode_src_k10d10_t` {  
`kClockTsiActiveSrcBusClk`,  
`kClockTsiActiveSrcMcgIrClk`,  
`kClockTsiActiveSrcOsc0erClk` }  
*TSI Active Mode clock source.*
- enum `clock_tsi_lp_mode_src_k10d10_t` {  
`kClockTsiLpSrcLpoClk`,  
`kClockTsiLpSrcEr32kClk` }  
*TSI Low-power Mode clock source.*
- enum `clock_pllfl_sel_k10d10_t` {  
`kClockPllFllSelFll` = 0U,  
`kClockPllFllSelPll` = 1U }  
*SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_k10d10_t` {  
`kClockEr32kSrcOsc0` = 0U,  
`kClockEr32kSrcRtc` = 2U,  
`kClockEr32kSrcLpo` = 3U }  
*SIM external reference clock source select (OSC32KSEL).*
- enum `clock_clkout_src_k10d10_t` {  
`kClockClkoutSelFlexbusClk` = 0U,  
`kClockClkoutSelFlashClk` = 2U,  
`kClockClkoutSelLpoClk` = 3U,  
`kClockClkoutSelMcgIrClk` = 4U,  
`kClockClkoutSelRtc32kClk` = 5U,  
`kClockClkoutSelOsc0erClk` = 6U }  
*SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_k10d10_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrc32kHz` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_k10d10_t` {  
`kSimAdcPretrgSelA`,

## SIM HAL driver

- kSimAdcPretrgSelB }
    - SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_k10d10_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_rxsrc_k10d10_t` {
  - kSimUartRxsrcPin,
  - kSimUartRxsrcCmp0,
  - kSimUartRxsrcCmp1 }
  - SIM UART receive data source select.*
- enum `sim_uart_txsrc_k10d10_t` {
  - kSimUartTxsrcPin,
  - kSimUartTxsrcFtm1,
  - kSimUartTxsrcFtm2 }
  - SIM UART transmit data source select.*
- enum `sim_ftm_trg_src_k10d10_t` {
  - kSimFtmTrgSrc0,
  - kSimFtmTrgSrc1 }
  - SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_k10d10_t` {
  - kSimFtmClkSel0,
  - kSimFtmClkSel1 }
  - SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_k10d10_t` {
  - kSimFtmChSrc0,
  - kSimFtmChSrc1,
  - kSimFtmChSrc2,
  - kSimFtmChSrc3 }
  - SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftmflt_sel_k10d10_t` {
  - kSimFtmFltSel0,
  - kSimFtmFltSel1 }
  - SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_k10d10_t` {

- `kSimTpmClkSel0,`
  - `kSimTpmClkSel1 }`

*SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_k10d10_t` {
  - `kSimTpmChSrc0,`
  - `kSimTpmChSrc1 }`

*SIM Timer/PWM x channel y input capture source select.*
- enum `sim_cmtuartpad_strenght_k10d10_t` {
  - `kSimCmtuartSinglePad,`
  - `kSimCmtuartDualPad }`

*SIM CMT/UART pad drive strength.*
- enum `sim_ptd7pad_strenght_k10d10_t` {
  - `kSimPtd7padSinglePad,`
  - `kSimPtd7padDualPad }`

*SIM PTD7 pad drive strength.*
- enum `sim_flexbus_security_level_k10d10_t` {
  - `kSimFbslLevel0,`
  - `kSimFbslLevel1,`
  - `kSimFbslLevel2,`
  - `kSimFbslLevel3 }`

*SIM FlexBus security level.*
- enum `sim_clock_gate_name_k10d10_t`

*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.12.2 Macro Definition Documentation

53.2.12.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.12.3 Enumeration Type Documentation

#### 53.2.12.3.1 enum `clock_wdog_src_k10d10_t`

Enumerator

- `kClockWdogSrcLpoClk` LPO.
- `kClockWdogSrcAltClk` Alternative clock, for K10D10 it is Bus clock.

#### 53.2.12.3.2 enum `clock_trace_src_k10d10_t`

Enumerator

- `kClockTraceSrcMcgoutClk` MCG out clock.
- `kClockTraceSrcCoreClk` core clock

## SIM HAL driver

### 53.2.12.3.3 enum clock\_port\_filter\_src\_k10d10\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.  
*kClockPortFilterSrcLpoClk* LPO.

### 53.2.12.3.4 enum clock\_lptmr\_src\_k10d10\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG IRC clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.12.3.5 enum clock\_time\_src\_k10d10\_t

Enumerator

*kClockTimeSrcCoreSysClk* Core/system clock.  
*kClockTimeSrcPllFllSel* clock as selected by SOPT2[PLLFLLSSEL].  
*kClockTimeSrcOsc0erClk* OSCERCLK clock.  
*kClockTimeSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

### 53.2.12.3.6 enum clock\_rmii\_src\_k10d10\_t

Enumerator

*kClockRmiiSrcExtalClk* EXTAL Clock.  
*kClockRmiiSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

### 53.2.12.3.7 enum clock\_flexcan\_src\_k10d10\_t

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.

### 53.2.12.3.8 enum clock\_sdhc\_src\_k10d10\_t

Enumerator

*kClockSdhcSrcCoreSysClk* Core/system clock.  
*kClockSdhcSrcPllFllSel* clock as selected by SOPT2[PLLFLLSEL].  
*kClockSdhcSrcOsc0erClk* OSCERCLK clock.  
*kClockSdhcSrcExt* External bypass clock (SDHC0\_CLKIN)

### 53.2.12.3.9 enum clock\_sai\_src\_k10d10\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.12.3.10 enum clock\_tsi\_active\_mode\_src\_k10d10\_t

Enumerator

*kClockTsiActiveSrcBusClk* Bus clock.  
*kClockTsiActiveSrcMcgIrClk* MCG IRC clock.  
*kClockTsiActiveSrcOsc0erClk* OSCERCLK clock.

### 53.2.12.3.11 enum clock\_tsi\_lp\_mode\_src\_k10d10\_t

Enumerator

*kClockTsiLpSrcLpoClk* LPO clock.  
*kClockTsiLpSrcEr32kClk* ERCLK32K clock.

### 53.2.12.3.12 enum clock\_pllfl\_sel\_k10d10\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.

## SIM HAL driver

### 53.2.12.3.13 enum clock\_er32k\_src\_k10d10\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.12.3.14 enum clock\_clkout\_src\_k10d10\_t

Enumerator

*kClockClkoutSelFlexbusClk* Flexbus clock.  
*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelRtc32kClk* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSCERCLK0 clock.

### 53.2.12.3.15 enum clock\_rtcout\_src\_k10d10\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

### 53.2.12.3.16 enum sim\_adc\_pretrg\_sel\_k10d10\_t

Enumerator

*kSimAdcPretrgselA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

### 53.2.12.3.17 enum sim\_adc\_trg\_sel\_k10d10\_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.

#### 53.2.12.3.18 enum sim\_uart\_rxsrc\_k10d10\_t

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.  
*kSimUartRxsrcCmp0* CMP0.  
*kSimUartRxsrcCmp1* CMP1.

#### 53.2.12.3.19 enum sim\_uart\_txsrc\_k10d10\_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.

#### 53.2.12.3.20 enum sim\_ftm\_trg\_src\_k10d10\_t

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.  
*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

#### 53.2.12.3.21 enum sim\_ftm\_clk\_sel\_k10d10\_t

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.  
*kSimFtmClkSel1* FTM CLKIN1 pin.

## SIM HAL driver

### 53.2.12.3.22 enum sim\_ftm\_ch\_src\_k10d10\_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.

### 53.2.12.3.23 enum sim\_ftmflt\_sel\_k10d10\_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

### 53.2.12.3.24 enum sim\_tpm\_clk\_sel\_k10d10\_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 53.2.12.3.25 enum sim\_tpm\_ch\_src\_k10d10\_t

Enumerator

- kSimTpmChSrc0* TPMx\_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

### 53.2.12.3.26 enum sim\_cmtuartpad\_streng\_h\_k10d10\_t

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.

### 53.2.12.3.27 enum sim\_ptd7pad\_streng\_h\_k10d10\_t

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

**53.2.12.3.28 enum sim\_flexbus\_security\_level\_k10d10\_t**

Enumerator

*kSimFbslLevel0* FlexBus security level 0.*kSimFbslLevel1* FlexBus security level 1.*kSimFbslLevel2* FlexBus security level 2.*kSimFbslLevel3* FlexBus security level 3.**53.2.12.3.29 enum sim\_clock\_gate\_name\_k10d10\_t**

## SIM HAL driver

### 53.2.13 K11DA5 SIM HAL driver

#### 53.2.13.1 Overview

The section describes the enumerations, macros and data structures for K11DA5 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MK11DA5.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_k11da5\\_t](#)  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_k11da5\\_t](#) {  
[kClockTraceSrcMcgoutClk](#),  
[kClockTraceSrcCoreClk](#) }  
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_k11da5\\_t](#) {  
[kClockPortFilterSrcBusClk](#),  
[kClockPortFilterSrcLpoClk](#) }  
*PORTx digital input filter clock source select.*
- enum [clock\\_lptmr\\_src\\_k11da5\\_t](#) {  
[kClockLptmrSrcMcgIrClk](#),  
[kClockLptmrSrcLpoClk](#),  
[kClockLptmrSrcEr32kClk](#),  
[kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_time\\_src\\_k11da5\\_t](#) {  
[kClockTimeSrcCoreSysClk](#),  
[kClockTimeSrcPIIFllSel](#),  
[kClockTimeSrcOsc0erClk](#),  
[kClockTimeSrcExt](#) }  
*SIM timestamp clock source.*
- enum [clock\\_usbfs\\_src\\_k11da5\\_t](#) {  
[kClockUsbfsSrcExt](#),  
[kClockUsbfsSrcPIIFllSel](#) }  
*SIM USB FS clock source.*
- enum [clock\\_sai\\_src\\_k11da5\\_t](#) {  
[kClockSaiSrcSysClk](#) = 0U,  
[kClockSaiSrcOsc0erClk](#) = 1U,

- `kClockSaiSrcPllClk = 3U }`
  - SAI clock source.*
- enum `clock_pllflr_sel_k11da5_t` {
  - `kClockPllFlrSelFlr = 0U,`
  - `kClockPllFlrSelPll = 1U }`
  - SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_k11da5_t` {
  - `kClockEr32kSrcOsc0 = 0U,`
  - `kClockEr32kSrcRtc = 2U,`
  - `kClockEr32kSrcLpo = 3U }`
  - SIM external reference clock source select (OSC32KSEL).*
- enum `clock_clkout_src_k11da5_t` {
  - `kClockClkoutSelFlashClk = 2U,`
  - `kClockClkoutSelLpoClk = 3U,`
  - `kClockClkoutSelMcgIrClk = 4U,`
  - `kClockClkoutSelRtc32kClk = 5U,`
  - `kClockClkoutSelOsc0erClk = 6U }`
  - SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_k11da5_t` {
  - `kClockRtcoutSrc1Hz,`
  - `kClockRtcoutSrc32kHz }`
  - SIM RTCCLKOUTSEL clock source select.*
- enum `sim_usbsstby_mode_k11da5_t` {
  - `kSimUsbsstbyNoRegulator,`
  - `kSimUsbsstbyWithRegulator }`
  - SIM USB voltage regulator in standby mode setting during stop modes.*
- enum `sim_usbvstby_mode_k11da5_t` {
  - `kSimUsbvstbyNoRegulator,`
  - `kSimUsbvstbyWithRegulator }`
  - SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.*
- enum `sim_adc_pretrg_sel_k11da5_t` {
  - `kSimAdcPretrgselA,`
  - `kSimAdcPretrgselB }`
  - SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_k11da5_t` {
  - `kSimAdcTrgselExt = 0U,`
  - `kSimAdcTrgSelHighSpeedComp0 = 1U,`
  - `kSimAdcTrgSelHighSpeedComp1 = 2U,`
  - `kSimAdcTrgSelPit0 = 4U,`
  - `kSimAdcTrgSelPit1 = 5U,`
  - `kSimAdcTrgSelPit2 = 6U,`
  - `kSimAdcTrgSelPit3 = 7U,`
  - `kSimAdcTrgSelFtm0 = 8U,`
  - `kSimAdcTrgSelFtm1 = 9U,`
  - `kSimAdcTrgSelFtm2 = 10U,`
  - `kSimAdcTrgSelRtcAlarm = 12U,`
  - `kSimAdcTrgSelRtcSec = 13U,`

## SIM HAL driver

- `kSimAdcTrgSellptimer = 14U }`
  - SIM ADCx trigger select.*
- enum `sim_uart_rxsrc_k11da5_t` {
  - `kSimUartRxsrcPin,`
  - `kSimUartRxsrcCmp0,`
  - `kSimUartRxsrcCmp1 }`
  - SIM UART receive data source select.*
- enum `sim_uart_txsrc_k11da5_t` {
  - `kSimUartTxsrcPin,`
  - `kSimUartTxsrcFtm1,`
  - `kSimUartTxsrcFtm2 }`
  - SIM UART transmit data source select.*
- enum `sim_ftm_trg_src_k11d5_t` {
  - `kSimFtmTrgSrc0,`
  - `kSimFtmTrgSrc1 }`
  - SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_k11da5_t` {
  - `kSimFtmClkSel0,`
  - `kSimFtmClkSel1 }`
  - SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_k11da5_t` {
  - `kSimFtmChSrc0,`
  - `kSimFtmChSrc1,`
  - `kSimFtmChSrc2,`
  - `kSimFtmChSrc3 }`
  - SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftmflt_sel_k11da5_t` {
  - `kSimFtmFltSel0,`
  - `kSimFtmFltSel1 }`
  - SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_k11da5_t` {
  - `kSimTpmClkSel0,`
  - `kSimTpmClkSel1 }`
  - SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_k11da5_t` {
  - `kSimTpmChSrc0,`
  - `kSimTpmChSrc1 }`
  - SIM Timer/PWM x channel y input capture source select.*
- enum `sim_cmtuartpad_strength_k11da5_t` {
  - `kSimCmtuartSinglePad,`
  - `kSimCmtuartDualPad }`
  - SIM CMT/UART pad drive strength.*
- enum `sim_ptd7pad_strength_k11da5_t` {
  - `kSimPtd7padSinglePad,`
  - `kSimPtd7padDualPad }`
  - SIM PTD7 pad drive strength.*
- enum `sim_clock_gate_name_k11da5_t`
  - Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.13.2 Macro Definition Documentation

53.2.13.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.13.3 Enumeration Type Documentation

#### 53.2.13.3.1 enum clock\_trace\_src\_k11da5\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

#### 53.2.13.3.2 enum clock\_port\_filter\_src\_k11da5\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.

*kClockPortFilterSrcLpoClk* LPO.

#### 53.2.13.3.3 enum clock\_lptmr\_src\_k11da5\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG IRC clock.

*kClockLptmrSrcLpoClk* LPO clock.

*kClockLptmrSrcEr32kClk* ERCLK32K clock.

*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

#### 53.2.13.3.4 enum clock\_time\_src\_k11da5\_t

Enumerator

*kClockTimeSrcCoreSysClk* Core/system clock.

*kClockTimeSrcPllFllSel* clock as selected by SOPT2[PLLFLLSEL].

*kClockTimeSrcOsc0erClk* OSCERCLK clock.

*kClockTimeSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

#### 53.2.13.3.5 enum clock\_usbfs\_src\_k11da5\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)

*kClockUsbfsSrcPllFllSel* Clock divider USB FS clock.

## SIM HAL driver

### 53.2.13.3.6 enum clock\_sai\_src\_k11da5\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.13.3.7 enum clock\_pllfl\_sel\_k11da5\_t

Enumerator

*kClockPllFlSelFl* Fll clock.  
*kClockPllFlSelPll* Pll0 clock.

### 53.2.13.3.8 enum clock\_er32k\_src\_k11da5\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.13.3.9 enum clock\_clkout\_src\_k11da5\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelRtc32kClk* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSCERCLK0 clock.

### 53.2.13.3.10 enum clock\_rtcout\_src\_k11da5\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock



**53.2.13.3.11 enum sim\_usbsstby\_mode\_k11da5\_t**

Enumerator

*kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes*kSimUsbsstbyWithRegulator* regulator in standby during Stop modes**53.2.13.3.12 enum sim\_usbvstby\_mode\_k11da5\_t**

Enumerator

*kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes*kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes**53.2.13.3.13 enum sim\_adc\_pretrg\_sel\_k11da5\_t**

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.**53.2.13.3.14 enum sim\_adc\_trg\_sel\_k11da5\_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.**53.2.13.3.15 enum sim\_uart\_rxsrc\_k11da5\_t**

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.

## SIM HAL driver

*kSimUartRxsrcCmp0* CMP0.

*kSimUartRxsrcCmp1* CMP1.

### 53.2.13.3.16 enum sim\_uart\_txsrc\_k11da5\_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.

### 53.2.13.3.17 enum sim\_ftm\_trg\_src\_k11d5\_t

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.

*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

### 53.2.13.3.18 enum sim\_ftm\_clk\_sel\_k11da5\_t

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.

*kSimFtmClkSel1* FTM CLKIN1 pin.

### 53.2.13.3.19 enum sim\_ftm\_ch\_src\_k11da5\_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.

### 53.2.13.3.20 enum sim\_ftmflt\_sel\_k11da5\_t

Enumerator

*kSimFtmFltSel0* FlexTimer x fault y select 0.

*kSimFtmFltSel1* FlexTimer x fault y select 1.

**53.2.13.3.21 enum sim\_tpm\_clk\_sel\_k11da5\_t**

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

**53.2.13.3.22 enum sim\_tpm\_ch\_src\_k11da5\_t**

Enumerator

- kSimTpmChSrc0* TPMx\_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

**53.2.13.3.23 enum sim\_cmtuartpad\_strength\_k11da5\_t**

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.

**53.2.13.3.24 enum sim\_ptd7pad\_strength\_k11da5\_t**

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

**53.2.13.3.25 enum sim\_clock\_gate\_name\_k11da5\_t**

## SIM HAL driver

### 53.2.14 K20D10 SIM HAL driver

#### 53.2.14.1 Overview

The section describes the enumerations, macros and data structures for K20D10 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MK20D10.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_k20d10\\_t](#) {  
[kClockWdogSrcLpoClk](#),  
[kClockWdogSrcAltClk](#) }  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_k20d10\\_t](#) {  
[kClockTraceSrcMcgoutClk](#),  
[kClockTraceSrcCoreClk](#) }  
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_k20d10\\_t](#) {  
[kClockPortFilterSrcBusClk](#),  
[kClockPortFilterSrcLpoClk](#) }  
*PORTx digital input filter clock source select.*
- enum [clock\\_lptmr\\_src\\_k20d10\\_t](#) {  
[kClockLptmrSrcMcgIrClk](#),  
[kClockLptmrSrcLpoClk](#),  
[kClockLptmrSrcEr32kClk](#),  
[kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_time\\_src\\_k20d10\\_t](#) {  
[kClockTimeSrcCoreSysClk](#),  
[kClockTimeSrcPIIFilSel](#),  
[kClockTimeSrcOsc0erClk](#),  
[kClockTimeSrcExt](#) }  
*SIM timestamp clock source.*
- enum [clock\\_rmii\\_src\\_k20d10\\_t](#) {  
[kClockRmiiSrcExtalClk](#),  
[kClockRmiiSrcExt](#) }  
*SIM RMII clock source.*

- enum `clock_flexcan_src_k20d10_t` {  
`kClockFlexcanSrcOsc0erClk`,  
`kClockFlexcanSrcBusClk` }  
*FLEXCAN clock source select.*
- enum `clock_sdhc_src_k20d10_t` {  
`kClockSdhcSrcCoreSysClk`,  
`kClockSdhcSrcPllFllSel`,  
`kClockSdhcSrcOsc0erClk`,  
`kClockSdhcSrcExt` }  
*SDHC clock source.*
- enum `clock_sai_src_k20d10_t` {  
`kClockSaiSrcSysClk` = 0U,  
`kClockSaiSrcOsc0erClk` = 1U,  
`kClockSaiSrcPllClk` = 3U }  
*SAI clock source.*
- enum `clock_tsi_active_mode_src_k20d10_t` {  
`kClockTsiActiveSrcBusClk`,  
`kClockTsiActiveSrcMcgIrClk`,  
`kClockTsiActiveSrcOsc0erClk` }  
*TSI Active Mode clock source.*
- enum `clock_tsi_lp_mode_src_k20d10_t` {  
`kClockTsiLpSrcLpoClk`,  
`kClockTsiLpSrcEr32kClk` }  
*TSI Low-power Mode clock source.*
- enum `clock_pllfl_sel_k20d10_t` {  
`kClockPllFllSelFll` = 0U,  
`kClockPllFllSelPll` = 1U }  
*SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_k20d10_t` {  
`kClockEr32kSrcOsc0` = 0U,  
`kClockEr32kSrcRtc` = 2U,  
`kClockEr32kSrcLpo` = 3U }  
*SIM external reference clock source select (OSC32KSEL).*
- enum `clock_clkout_src_k20d10_t` {  
`kClockClkoutSelFlexbusClk` = 0U,  
`kClockClkoutSelFlashClk` = 2U,  
`kClockClkoutSelLpoClk` = 3U,  
`kClockClkoutSelMcgIrClk` = 4U,  
`kClockClkoutSelRtc32kClk` = 5U,  
`kClockClkoutSelOsc0erClk` = 6U }  
*SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_k20d10_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrc32kHz` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_k20d10_t` {  
`kSimAdcPretrgSelA`,

## SIM HAL driver

- kSimAdcPretrgSelB }
    - SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_k20d10_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_rxsrc_k20d10_t` {
  - kSimUartRxsrcPin,
  - kSimUartRxsrcCmp0,
  - kSimUartRxsrcCmp1 }
  - SIM UART receive data source select.*
- enum `sim_uart_txsrc_k20d10_t` {
  - kSimUartTxsrcPin,
  - kSimUartTxsrcFtm1,
  - kSimUartTxsrcFtm2 }
  - SIM UART transmit data source select.*
- enum `sim_ftm_trg_src_k20d10_t` {
  - kSimFtmTrgSrc0,
  - kSimFtmTrgSrc1 }
  - SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_k20d10_t` {
  - kSimFtmClkSel0,
  - kSimFtmClkSel1 }
  - SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_k20d10_t` {
  - kSimFtmChSrc0,
  - kSimFtmChSrc1,
  - kSimFtmChSrc2,
  - kSimFtmChSrc3 }
  - SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftmflt_sel_k20d10_t` {
  - kSimFtmFltSel0,
  - kSimFtmFltSel1 }
  - SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_k20d10_t` {

- kSimTpmClkSel0,
  - kSimTpmClkSel1 }
    - SIM Timer/PWM external clock select.*
  - enum `sim_tpm_ch_src_k20d10_t` {
    - kSimTpmChSrc0,
    - kSimTpmChSrc1 }
      - SIM Timer/PWM x channel y input capture source select.*
    - enum `sim_cmtuartpad_streng_h_k20d10_t` {
      - kSimCmtuartSinglePad,
      - kSimCmtuartDualPad }
        - SIM CMT/UART pad drive strength.*
      - enum `sim_ptd7pad_streng_h_k20d10_t` {
        - kSimPtd7padSinglePad,
        - kSimPtd7padDualPad }
          - SIM PTD7 pad drive strength.*
        - enum `sim_flexbus_security_level_k20d10_t` {
          - kSimFbslLevel0,
          - kSimFbslLevel1,
          - kSimFbslLevel2,
          - kSimFbslLevel3 }
            - SIM FlexBus security level.*
          - enum `sim_clock_gate_name_k20d10_t`
            - Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.14.2 Macro Definition Documentation

53.2.14.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.14.3 Enumeration Type Documentation

#### 53.2.14.3.1 enum `clock_wdog_src_k20d10_t`

Enumerator

- kClockWdogSrcLpoClk* LPO.
- kClockWdogSrcAltClk* Alternative clock, for K20D10 it is Bus clock.

#### 53.2.14.3.2 enum `clock_trace_src_k20d10_t`

Enumerator

- kClockTraceSrcMcgoutClk* MCG out clock.
- kClockTraceSrcCoreClk* core clock

## SIM HAL driver

### 53.2.14.3.3 enum clock\_port\_filter\_src\_k20d10\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.  
*kClockPortFilterSrcLpoClk* LPO.

### 53.2.14.3.4 enum clock\_lptmr\_src\_k20d10\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG IRC clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.14.3.5 enum clock\_time\_src\_k20d10\_t

Enumerator

*kClockTimeSrcCoreSysClk* Core/system clock.  
*kClockTimeSrcPllFllSel* clock as selected by SOPT2[PLLFLLSSEL].  
*kClockTimeSrcOsc0erClk* OSCERCLK clock.  
*kClockTimeSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

### 53.2.14.3.6 enum clock\_rmii\_src\_k20d10\_t

Enumerator

*kClockRmiiSrcExtalClk* EXTAL Clock.  
*kClockRmiiSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

### 53.2.14.3.7 enum clock\_flexcan\_src\_k20d10\_t

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.



### 53.2.14.3.8 enum clock\_sdhc\_src\_k20d10\_t

Enumerator

*kClockSdhcSrcCoreSysClk* Core/system clock.  
*kClockSdhcSrcPllFllSel* clock as selected by SOPT2[PLLFLLSEL].  
*kClockSdhcSrcOsc0erClk* OSCERCLK clock.  
*kClockSdhcSrcExt* External bypass clock (SDHC0\_CLKIN)

### 53.2.14.3.9 enum clock\_sai\_src\_k20d10\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.14.3.10 enum clock\_tsi\_active\_mode\_src\_k20d10\_t

Enumerator

*kClockTsiActiveSrcBusClk* Bus clock.  
*kClockTsiActiveSrcMcgIrClk* MCG IRC clock.  
*kClockTsiActiveSrcOsc0erClk* OSCERCLK clock.

### 53.2.14.3.11 enum clock\_tsi\_lp\_mode\_src\_k20d10\_t

Enumerator

*kClockTsiLpSrcLpoClk* LPO clock.  
*kClockTsiLpSrcEr32kClk* ERCLK32K clock.

### 53.2.14.3.12 enum clock\_pllfl\_sel\_k20d10\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.

## SIM HAL driver

### 53.2.14.3.13 enum clock\_er32k\_src\_k20d10\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.14.3.14 enum clock\_clkout\_src\_k20d10\_t

Enumerator

*kClockClkoutSelFlexbusClk* Flexbus clock.  
*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelRtc32kClk* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSCERCLK0 clock.

### 53.2.14.3.15 enum clock\_rtcout\_src\_k20d10\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

### 53.2.14.3.16 enum sim\_adc\_pretrg\_sel\_k20d10\_t

Enumerator

*kSimAdcPretrgselA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

### 53.2.14.3.17 enum sim\_adc\_trg\_sel\_k20d10\_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.

#### 53.2.14.3.18 enum sim\_uart\_rxsrc\_k20d10\_t

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.  
*kSimUartRxsrcCmp0* CMP0.  
*kSimUartRxsrcCmp1* CMP1.

#### 53.2.14.3.19 enum sim\_uart\_txsrc\_k20d10\_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.

#### 53.2.14.3.20 enum sim\_ftm\_trg\_src\_k20d10\_t

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.  
*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

#### 53.2.14.3.21 enum sim\_ftm\_clk\_sel\_k20d10\_t

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.  
*kSimFtmClkSel1* FTM CLKIN1 pin.

## SIM HAL driver

### 53.2.14.3.22 enum sim\_ftm\_ch\_src\_k20d10\_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.

### 53.2.14.3.23 enum sim\_ftmflt\_sel\_k20d10\_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

### 53.2.14.3.24 enum sim\_tpm\_clk\_sel\_k20d10\_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 53.2.14.3.25 enum sim\_tpm\_ch\_src\_k20d10\_t

Enumerator

- kSimTpmChSrc0* TPMx\_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

### 53.2.14.3.26 enum sim\_cmtuartpad\_strength\_k20d10\_t

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.

### 53.2.14.3.27 enum sim\_ptd7pad\_strength\_k20d10\_t

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

**53.2.14.3.28 enum sim\_flexbus\_security\_level\_k20d10\_t**

Enumerator

*kSimFbslLevel0* FlexBus security level 0.*kSimFbslLevel1* FlexBus security level 1.*kSimFbslLevel2* FlexBus security level 2.*kSimFbslLevel3* FlexBus security level 3.**53.2.14.3.29 enum sim\_clock\_gate\_name\_k20d10\_t**

## SIM HAL driver

### 53.2.15 K21DA5 SIM HAL driver

#### 53.2.15.1 Overview

The section describes the enumerations, macros and data structures for K21DA5 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MK21DA5.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_k21da5\\_t](#)  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_k21da5\\_t](#) {  
[kClockTraceSrcMcgoutClk](#),  
[kClockTraceSrcCoreClk](#) }  
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_k21da5\\_t](#) {  
[kClockPortFilterSrcBusClk](#),  
[kClockPortFilterSrcLpoClk](#) }  
*PORTx digital input filter clock source select.*
- enum [clock\\_lptmr\\_src\\_k21da5\\_t](#) {  
[kClockLptmrSrcMcgIrClk](#),  
[kClockLptmrSrcLpoClk](#),  
[kClockLptmrSrcEr32kClk](#),  
[kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_time\\_src\\_k21da5\\_t](#) {  
[kClockTimeSrcCoreSysClk](#),  
[kClockTimeSrcPIIFllSel](#),  
[kClockTimeSrcOsc0erClk](#),  
[kClockTimeSrcExt](#) }  
*SIM timestamp clock source.*
- enum [clock\\_usbfs\\_src\\_k21da5\\_t](#) {  
[kClockUsbfsSrcExt](#),  
[kClockUsbfsSrcPIIFllSel](#) }  
*SIM USB FS clock source.*
- enum [clock\\_sai\\_src\\_k21da5\\_t](#) {  
[kClockSaiSrcSysClk](#) = 0U,  
[kClockSaiSrcOsc0erClk](#) = 1U,

- kClockSaiSrcPllClk = 3U }
  - SAI clock source.*
- enum clock\_pllfl1\_sel\_k21da5\_t {
  - kClockPllFl1SelFl1 = 0U,
  - kClockPllFl1SelPll = 1U }
  - SIM PLLFLLSEL clock source select.*
- enum clock\_er32k\_src\_k21da5\_t {
  - kClockEr32kSrcOsc0 = 0U,
  - kClockEr32kSrcRtc = 2U,
  - kClockEr32kSrcLpo = 3U }
  - SIM external reference clock source select (OSC32KSEL).*
- enum clock\_clkout\_src\_k21da5\_t {
  - kClockClkoutSelFlashClk = 2U,
  - kClockClkoutSelLpoClk = 3U,
  - kClockClkoutSelMcgIrClk = 4U,
  - kClockClkoutSelRtc32kClk = 5U,
  - kClockClkoutSelOsc0erClk = 6U }
  - SIM CLKOUT\_SEL clock source select.*
- enum clock\_rtcout\_src\_k21da5\_t {
  - kClockRtcoutSrc1Hz,
  - kClockRtcoutSrc32kHz }
  - SIM RTCCLKOUTSEL clock source select.*
- enum sim\_usbsstby\_mode\_k21da5\_t {
  - kSimUsbsstbyNoRegulator,
  - kSimUsbsstbyWithRegulator }
  - SIM USB voltage regulator in standby mode setting during stop modes.*
- enum sim\_usbvstby\_mode\_k21da5\_t {
  - kSimUsbvstbyNoRegulator,
  - kSimUsbvstbyWithRegulator }
  - SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.*
- enum sim\_adc\_pretrg\_sel\_k21da5\_t {
  - kSimAdcPretrgselA,
  - kSimAdcPretrgselB }
  - SIM ADCx pre-trigger select.*
- enum sim\_adc\_trg\_sel\_k21da5\_t {
  - kSimAdcTrgselExt = 0U,
  - kSimAdcTrgSelHighSpeedComp0 = 1U,
  - kSimAdcTrgSelHighSpeedComp1 = 2U,
  - kSimAdcTrgSelPit0 = 4U,
  - kSimAdcTrgSelPit1 = 5U,
  - kSimAdcTrgSelPit2 = 6U,
  - kSimAdcTrgSelPit3 = 7U,
  - kSimAdcTrgSelFtm0 = 8U,
  - kSimAdcTrgSelFtm1 = 9U,
  - kSimAdcTrgSelFtm2 = 10U,
  - kSimAdcTrgSelRtcAlarm = 12U,
  - kSimAdcTrgSelRtcSec = 13U,

## SIM HAL driver

- `kSimAdcTrgSellptimer = 14U }`
  - SIM ADCx trigger select.*
- enum `sim_uart_rxsrc_k21da5_t` {
  - `kSimUartRxsrcPin,`
  - `kSimUartRxsrcCmp0,`
  - `kSimUartRxsrcCmp1 }`
  - SIM UART receive data source select.*
- enum `sim_uart_txsrc_k21da5_t` {
  - `kSimUartTxsrcPin,`
  - `kSimUartTxsrcFtm1,`
  - `kSimUartTxsrcFtm2 }`
  - SIM UART transmit data source select.*
- enum `sim_ftm_trg_src_k21da5_t` {
  - `kSimFtmTrgSrc0,`
  - `kSimFtmTrgSrc1 }`
  - SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_k21da5_t` {
  - `kSimFtmClkSel0,`
  - `kSimFtmClkSel1 }`
  - SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_k21da5_t` {
  - `kSimFtmChSrc0,`
  - `kSimFtmChSrc1,`
  - `kSimFtmChSrc2,`
  - `kSimFtmChSrc3 }`
  - SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftmflt_sel_k21da5_t` {
  - `kSimFtmFltSel0,`
  - `kSimFtmFltSel1 }`
  - SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_k21da5_t` {
  - `kSimTpmClkSel0,`
  - `kSimTpmClkSel1 }`
  - SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_k21da5_t` {
  - `kSimTpmChSrc0,`
  - `kSimTpmChSrc1 }`
  - SIM Timer/PWM x channel y input capture source select.*
- enum `sim_cmtuartpad_strength_k21da5_t` {
  - `kSimCmtuartSinglePad,`
  - `kSimCmtuartDualPad }`
  - SIM CMT/UART pad drive strength.*
- enum `sim_ptd7pad_strength_k21da5_t` {
  - `kSimPtd7padSinglePad,`
  - `kSimPtd7padDualPad }`
  - SIM PTD7 pad drive strength.*
- enum `sim_clock_gate_name_k21da5_t`
  - Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*



### 53.2.15.2 Macro Definition Documentation

53.2.15.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.15.3 Enumeration Type Documentation

#### 53.2.15.3.1 enum clock\_trace\_src\_k21da5\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

#### 53.2.15.3.2 enum clock\_port\_filter\_src\_k21da5\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.

*kClockPortFilterSrcLpoClk* LPO.

#### 53.2.15.3.3 enum clock\_lptmr\_src\_k21da5\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG IRC clock.

*kClockLptmrSrcLpoClk* LPO clock.

*kClockLptmrSrcEr32kClk* ERCLK32K clock.

*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

#### 53.2.15.3.4 enum clock\_time\_src\_k21da5\_t

Enumerator

*kClockTimeSrcCoreSysClk* Core/system clock.

*kClockTimeSrcPllFllSel* clock as selected by SOPT2[PLLFLLSSEL].

*kClockTimeSrcOsc0erClk* OSCERCLK clock.

*kClockTimeSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

#### 53.2.15.3.5 enum clock\_usbfs\_src\_k21da5\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)

*kClockUsbfsSrcPllFllSel* Clock divider USB FS clock.

## SIM HAL driver

### 53.2.15.3.6 enum clock\_sai\_src\_k21da5\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.15.3.7 enum clock\_pllfl\_sel\_k21da5\_t

Enumerator

*kClockPllFlSelFl* Fll clock.  
*kClockPllFlSelPll* Pll0 clock.

### 53.2.15.3.8 enum clock\_er32k\_src\_k21da5\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.15.3.9 enum clock\_clkout\_src\_k21da5\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelRtc32kClk* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSCERCLK0 clock.

### 53.2.15.3.10 enum clock\_rtcout\_src\_k21da5\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

### 53.2.15.3.11 enum sim\_usbsstby\_mode\_k21da5\_t

Enumerator

*kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes

*kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

### 53.2.15.3.12 enum sim\_usbvstby\_mode\_k21da5\_t

Enumerator

*kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes

*kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes

### 53.2.15.3.13 enum sim\_adc\_pretrg\_sel\_k21da5\_t

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.

*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.15.3.14 enum sim\_adc\_trg\_sel\_k21da5\_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.

### 53.2.15.3.15 enum sim\_uart\_rxsrc\_k21da5\_t

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.

## SIM HAL driver

*kSimUartRxsrcCmp0* CMP0.

*kSimUartRxsrcCmp1* CMP1.

### 53.2.15.3.16 enum sim\_uart\_txsrc\_k21da5\_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.

### 53.2.15.3.17 enum sim\_ftm\_trg\_src\_k21da5\_t

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.

*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

### 53.2.15.3.18 enum sim\_ftm\_clk\_sel\_k21da5\_t

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.

*kSimFtmClkSel1* FTM CLKIN1 pin.

### 53.2.15.3.19 enum sim\_ftm\_ch\_src\_k21da5\_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.

### 53.2.15.3.20 enum sim\_ftmflt\_sel\_k21da5\_t

Enumerator

*kSimFtmFltSel0* FlexTimer x fault y select 0.

*kSimFtmFltSel1* FlexTimer x fault y select 1.

**53.2.15.3.21 enum sim\_tpm\_clk\_sel\_k21da5\_t**

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

**53.2.15.3.22 enum sim\_tpm\_ch\_src\_k21da5\_t**

Enumerator

- kSimTpmChSrc0* TPMx\_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

**53.2.15.3.23 enum sim\_cmtuartpad\_strength\_k21da5\_t**

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.

**53.2.15.3.24 enum sim\_ptd7pad\_strength\_k21da5\_t**

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

**53.2.15.3.25 enum sim\_clock\_gate\_name\_k21da5\_t**

## SIM HAL driver

### 53.2.16 KL02Z4 SIM HAL driver

#### 53.2.16.1 Overview

The section describes the enumerations, macros and data structures for KL02Z4 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKL02Z4.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_cop\\_src\\_kl02z4\\_t](#) {  
  [kClockCopSrcLpoClk](#),  
  [kClockCopSrcAltClk](#) }  
*COP clock source select.*
- enum [clock\\_tpm\\_src\\_kl02z4\\_t](#) {  
  [kClockTpmSrcNone](#),  
  [kClockTpmSrcFll](#),  
  [kClockTpmSrcOsc0erClk](#),  
  [kClockTpmSrcMcgIrClk](#) }  
*TPM clock source select.*
- enum [clock\\_lptmr\\_src\\_kl02z4\\_t](#) {  
  [kClockLptmrSrcMcgIrClk](#),  
  [kClockLptmrSrcLpoClk](#),  
  [kClockLptmrSrcEr32kClk](#),  
  [kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_lpsci\\_src\\_kl02z4\\_t](#) {  
  [kClockLpsciSrcNone](#),  
  [kClockLpsciSrcFll](#),  
  [kClockLpsciSrcOsc0erClk](#),  
  [kClockLpsciSrcMcgIrClk](#) }  
*UART0 clock source select.*
- enum [clock\\_clkout\\_src\\_kl02z4\\_t](#) {

```

kClockClkoutReserved = 0U,
kClockClkoutReserved1 = 1U,
kClockClkoutBusClk = 2U,
kClockClkoutLpoClk = 3U,
kClockClkoutMcgIrClk = 4U,
kClockClkoutReserved2 = 5U,
kClockClkoutOsc0erClk = 6U,
kClockClkoutReserved3 = 7U }

```

*SIM CLKOUT\_SEL clock source select.*

- enum `sim_adc_pretrg_sel_kl02z4_t` {  
`kSimAdcPretrgselA`,  
`kSimAdcPretrgselB` }

*SIM ADCx pre-trigger select.*

- enum `sim_adc_trg_sel_kl02z4_t` {  
`kSimAdcTrgselExt` = 0U,  
`kSimAdcTrgSelComp0` = 1U,  
`kSimAdcTrgSelReserved2` = 2U,  
`kSimAdcTrgSelReserved3` = 3U,  
`kSimAdcTrgSelReserved4` = 4U,  
`kSimAdcTrgSelReserved5` = 5U,  
`kSimAdcTrgSelReserved6` = 6U,  
`kSimAdcTrgSelReserved7` = 7U,  
`kSimAdcTrgSelTpm0` = 8U,  
`kSimAdcTrgSelTpm1` = 9U,  
`kSimAdcTrgSelReserved10` = 10U,  
`kSimAdcTrgSelReserved11` = 11U,  
`kSimAdcTrgSelReserved12` = 12U,  
`kSimAdcTrgSelReserved13` = 13U,  
`kSimAdcTrgSelLptimer` = 14U,  
`kSimAdcTrgSelReserved15` = 15U }

*SIM ADCx trigger select.*

- enum `sim_lpsci_rxsrc_kl02z4_t` {  
`kSimLpsciRxsrcPin`,  
`kSimLpsciRxsrcCmp0` }

*SIM LPSCI receive data source select.*

- enum `sim_lpsci_txsrc_kl02z4_t` {  
`kSimLpsciTxsrcPin`,  
`kSimLpsciTxsrcTpm1` }

*SIM LPSCI transmit data source select.*

- enum `sim_tpm_clk_sel_kl02z4_t` {  
`kSimTpmClkSel0`,  
`kSimTpmClkSel1` }

*SIM Timer/PWM external clock select.*

- enum `sim_tpm_ch_src_kl02z4_t` {  
`kSimTpmChSrc0`,  
`kSimTpmChSrc1` }

## SIM HAL driver

- SIM Timer/PWM x channel y input capture source select.*

  - enum `sim_clock_gate_name_kl02z4_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.16.2 Macro Definition Documentation

53.2.16.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.16.3 Enumeration Type Documentation

53.2.16.3.1 enum `clock_cop_src_kl02z4_t`

Enumerator

*kClockCopSrcLpoClk* LPO.  
*kClockCopSrcAltClk* Alternative clock, for KL02Z4 it is Bus clock.

53.2.16.3.2 enum `clock_tpm_src_kl02z4_t`

Enumerator

*kClockTpmSrcNone* clock disabled  
*kClockTpmSrcFll* MCGFLLCLK.  
*kClockTpmSrcOsc0erClk* OSCERCLK clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

53.2.16.3.3 enum `clock_lptmr_src_kl02z4_t`

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

53.2.16.3.4 enum `clock_lpsci_src_kl02z4_t`

Enumerator

*kClockLpsciSrcNone* clock disabled  
*kClockLpsciSrcFll* MCGFLLCLK.  
*kClockLpsciSrcOsc0erClk* OSCERCLK clock.  
*kClockLpsciSrcMcgIrClk* MCGIR clock.



**53.2.16.3.5 enum clock\_clkout\_src\_kl02z4\_t**

Enumerator

*kClockClkoutReserved* Reserved.  
*kClockClkoutReserved1* Reserved.  
*kClockClkoutBusClk* Bus clock.  
*kClockClkoutLpoClk* LPO clock.  
*kClockClkoutMcgIrClk* MCG ir clock.  
*kClockClkoutReserved2* Reserved.  
*kClockClkoutOsc0erClk* OSC0ER clock.  
*kClockClkoutReserved3* Reserved.

**53.2.16.3.6 enum sim\_adc\_pretrg\_sel\_kl02z4\_t**

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

**53.2.16.3.7 enum sim\_adc\_trg\_sel\_kl02z4\_t**

Enumerator

*kSimAdcTrgSelExt* External trigger.  
*kSimAdcTrgSelComp0* CMP0 output.  
*kSimAdcTrgSelReserved2* Reserved.  
*kSimAdcTrgSelReserved3* Reserved.  
*kSimAdcTrgSelReserved4* Reserved.  
*kSimAdcTrgSelReserved5* Reserved.  
*kSimAdcTrgSelReserved6* Reserved.  
*kSimAdcTrgSelReserved7* Reserved.  
*kSimAdcTrgSelTpm0* TPM0 overflow.  
*kSimAdcTrgSelTpm1* TPM1 overflow.  
*kSimAdcTrgSelReserved10* Reserved.  
*kSimAdcTrgSelReserved11* Reserved.  
*kSimAdcTrgSelReserved12* Reserved.  
*kSimAdcTrgSelReserved13* Reserved.  
*kSimAdcTrgSelLptimer* Low-power timer trigger.  
*kSimAdcTrgSelReserved15* Reserved.

## SIM HAL driver

### 53.2.16.3.8 enum sim\_lpsci\_rxsrc\_kl02z4\_t

Enumerator

*kSimLpsciRxsrcPin* LPSCIx\_RX Pin.

*kSimLpsciRxsrcCmp0* CMP0.

### 53.2.16.3.9 enum sim\_lpsci\_txsrc\_kl02z4\_t

Enumerator

*kSimLpsciTxsrcPin* LPSCIx\_TX Pin.

*kSimLpsciTxsrcTpm1* LPSCIx\_TX pin modulated with TPM1 channel 0 output.

### 53.2.16.3.10 enum sim\_tpm\_clk\_sel\_kl02z4\_t

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.

*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 53.2.16.3.11 enum sim\_tpm\_ch\_src\_kl02z4\_t

Enumerator

*kSimTpmChSrc0* TPMx\_CH0 signal.

*kSimTpmChSrc1* CMP0 output.

### 53.2.16.3.12 enum sim\_clock\_gate\_name\_kl02z4\_t

## 53.2.17 KL03Z4 SIM HAL driver

### 53.2.17.1 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.*

## SIM HAL driver

- 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_lpuart_rxsrc_kl03z4_t` {  
`kSimLpuartRxsrcPin`,  
`kSimLpuartRxsrcCmp0` }  
*SIM LPUART receive data source select.*
- enum `sim_lpuart_txsrc_kl03z4_t` {  
`kSimLpuartTxsrcPin`,  
`kSimLpuartTxsrcFtm1` }  
*SIM LPUART 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 Timer/PWM x channel y input capture source select.*
- enum `sim_ptd7pad_streng_h_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.*

## 53.2.17.2 Macro Definition Documentation

53.2.17.2.1 #define [FSL\\_SIM\\_SCGC\\_BIT\( SCGCx, n \)](#) (((SCGCx-1U)<<5U) + n)

## 53.2.17.3 Enumeration Type Documentation

### 53.2.17.3.1 enum [clock\\_cop\\_src\\_kl03z4\\_t](#)

Enumerator

*kClockCopSrcLpoClk* LPO clock,1K HZ.  
*kClockCopSrcMcgIrClk* MCG IRC Clock.  
*kClockCopSrcOsc0erClk* OSCER Clock.  
*kClockCopSrcBusClk* BUS clock.

### 53.2.17.3.2 enum [clock\\_er32k\\_src\\_kl03z4\\_t](#)

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.17.3.3 enum [clock\\_osc32kout\\_sel\\_kl03z4\\_t](#)

Enumerator

*kClockOsc32koutNone* ERCLK32K is not output.  
*kClockOsc32koutPtB13* ERCLK32K output on PTB13.

### 53.2.17.3.4 enum [clock\\_lpuart\\_src\\_kl03z4\\_t](#)

Enumerator

*kClockLpuartSrcNone* disabled  
*kClockLpuartSrcIrc48M* IRC48M.  
*kClockLpuartSrcOsc0erClk* OSCER clock.  
*kClockLpuartSrcMcgIrClk* MCGIR clock.

## SIM HAL driver

### 53.2.17.3.5 enum clock\_tpm\_src\_kl03z4\_t

Enumerator

*kClockTpmSrcNone* disabled  
*kClockTpmSrcIrc48M* IRC48M.  
*kClockTpmSrcOsc0erClk* OSCER clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.17.3.6 enum clock\_lptmr\_src\_kl03z4\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.17.3.7 enum clock\_clkout\_src\_kl03z4\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelOsc0erClk* OSCER clock.  
*kClockClkoutSelIrc48M* IRC48M clock.

### 53.2.17.3.8 enum clock\_rtcout\_src\_kl03z4\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrcOsc0erClk* OSCER clock.

### 53.2.17.3.9 enum sim\_adc\_pretrg\_sel\_kl03z4\_t

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

**53.2.17.3.10 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.

**53.2.17.3.11 enum sim\_lpuart\_rxsrc\_kl03z4\_t**

Enumerator

- kSimLpuartRxsrcPin* LPUARTx\_RX Pin.
- kSimLpuartRxsrcCmp0* CMP0.

**53.2.17.3.12 enum sim\_lpuart\_txsrc\_kl03z4\_t**

Enumerator

- kSimLpuartTxsrcPin* LPUARTx\_TX Pin.
- kSimLpuartTxsrcFtm1* LPUARTx\_TX pin modulated with FTM1 channel 0 output.

**53.2.17.3.13 enum sim\_tpm\_clk\_sel\_kl03z4\_t**

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

**53.2.17.3.14 enum sim\_tpm\_ch\_src\_kl03z4\_t**

Enumerator

- kSimTpmChSrc0* TPMx\_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

## SIM HAL driver

### 53.2.17.3.15 enum sim\_ptd7pad\_strength\_kl03z4\_t

Enumerator

*kSimPtd7padSinglePad* Single-pad drive strength for PTD7.

*kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

### 53.2.17.3.16 enum sim\_clock\_gate\_name\_kl03z4\_t



### **53.2.18 KL13Z644 SIM HAL driver**

The section describes the enumerations, macros and data structures for KL13Z644 SIM HAL driver.

## SIM HAL driver

### 53.2.19 KL16Z4 SIM HAL driver

#### 53.2.19.1 Overview

The section describes the enumerations, macros and data structures for KL16Z4 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKL16Z4.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_cop\\_src\\_kl16z4\\_t](#) {  
  [kClockCopSrcLpoClk](#),  
  [kClockCopSrcAltClk](#) }  
*COP clock source select.*
- enum [clock\\_tpm\\_src\\_kl16z4\\_t](#) {  
  [kClockTpmSrcNone](#),  
  [kClockTpmSrcPllFllSel](#),  
  [kClockTpmSrcOsc0erClk](#),  
  [kClockTpmSrcMcgIrClk](#) }  
*TPM clock source select.*
- enum [clock\\_lptmr\\_src\\_kl16z4\\_t](#) {  
  [kClockLptmrSrcMcgIrClk](#),  
  [kClockLptmrSrcLpoClk](#),  
  [kClockLptmrSrcEr32kClk](#),  
  [kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_lpsci\\_src\\_kl16z4\\_t](#) {  
  [kClockLpsciSrcNone](#),  
  [kClockLpsciSrcPllFllSel](#),  
  [kClockLpsciSrcOsc0erClk](#),  
  [kClockLpsciSrcMcgIrClk](#) }  
*UART0 clock source select.*
- enum [clock\\_sai\\_src\\_kl16z4\\_t](#) {  
  [kClockSaiSrcSysClk](#) = 0U,  
  [kClockSaiSrcOsc0erClk](#) = 1U,  
  [kClockSaiSrcPllClk](#) = 3U }  
*SAI clock source.*

- enum `clock_pllfl1_sel_kl16z4_t` {  
`kClockPllFl1SelFl1`,  
`kClockPllFl1SelPll` }  
*SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_kl16z4_t` {  
`kClockEr32kSrcOsc0` = 0U,  
`kClockEr32kSrcReserved` = 1U,  
`kClockEr32kSrcRtc` = 2U,  
`kClockEr32kSrcLpo` = 3U }  
*SIM external reference clock source select (OSC32KSEL)*
- enum `clock_clkout_src_kl16z4_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_kl16z4_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrc32kHz` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_kl16z4_t` {  
`kSimAdcPretrgselA`,  
`kSimAdcPretrgselB` }  
*SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kl16z4_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.*

## SIM HAL driver

- enum `sim_uart_rxsrc_kl16z4_t` {  
`kSimUartRxsrcPin`,  
`kSimUartRxsrcCmp0` }  
*SIM UART receive data source select.*
- enum `sim_uart_txsrc_kl16z4_t` {  
`kSimUartTxsrcPin`,  
`kSimUartTxsrcTpm1`,  
`kSimUartTxsrcTpm2`,  
`kSimUartTxsrcReserved` }  
*SIM UART transmit data source select.*
- enum `sim_lpsci_rxsrc_kl16z4_t` {  
`kSimLpsciRxsrcPin`,  
`kSimLpsciRxsrcCmp0` }  
*SIM LPSCI receive data source select.*
- enum `sim_lpsci_txsrc_kl16z4_t` {  
`kSimLpsciTxsrcPin`,  
`kSimLpsciTxsrcTpm1`,  
`kSimLpsciTxsrcTpm2`,  
`kSimLpsciTxsrcReserved` }  
*SIM LPSCI transmit data source select.*
- enum `sim_tpm_clk_sel_kl16z4_t` {  
`kSimTpmClkSel0`,  
`kSimTpmClkSel1` }  
*SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_kl16z4_t` {  
`kSimTpmChSrc0`,  
`kSimTpmChSrc1`,  
`kSimTpmChSrc2`,  
`kSimTpmChSrc3` }  
*SIM Timer/PWM x channel y input capture source select.*
- enum `sim_clock_gate_name_kl16z4_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.19.2 Macro Definition Documentation

53.2.19.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.19.3 Enumeration Type Documentation

53.2.19.3.1 `enum clock_cop_src_kl16z4_t`

Enumerator

*kClockCopSrcLpoClk* LPO.

*kClockCopSrcAltClk* Alternative clock, for KL16Z4 it is Bus clock.

### 53.2.19.3.2 enum clock\_tpm\_src\_kl16z4\_t

Enumerator

*kClockTpmSrcNone* clock disabled  
*kClockTpmSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockTpmSrcOsc0erClk* OSCERCLK clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.19.3.3 enum clock\_lptmr\_src\_kl16z4\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.19.3.4 enum clock\_lpsci\_src\_kl16z4\_t

Enumerator

*kClockLpsciSrcNone* clock disabled  
*kClockLpsciSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockLpsciSrcOsc0erClk* OSCERCLK clock.  
*kClockLpsciSrcMcgIrClk* MCGIR clock.

### 53.2.19.3.5 enum clock\_sai\_src\_kl16z4\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.19.3.6 enum clock\_pllfl\_sel\_kl16z4\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.

## SIM HAL driver

### 53.2.19.3.7 enum clock\_er32k\_src\_kl16z4\_t

Enumerator

*kClockEr32kSrcOsc0* OSC 32k clock.  
*kClockEr32kSrcReserved* Reserved.  
*kClockEr32kSrcRtc* RTC 32k clock.  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.19.3.8 enum clock\_clkout\_src\_kl16z4\_t

Enumerator

*kClockClkoutReserved* Reserved.  
*kClockClkoutReserved1* Reserved.  
*kClockClkoutBusClk* Bus clock.  
*kClockClkoutLpoClk* LPO clock.  
*kClockClkoutMcgIrClk* MCG ir clock.  
*kClockClkoutReserved2* Reserved.  
*kClockClkoutOsc0erClk* OSC0ER clock.  
*kClockClkoutReserved3* Reserved.

### 53.2.19.3.9 enum clock\_rtcout\_src\_kl16z4\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32KHz clock

### 53.2.19.3.10 enum sim\_adc\_pretrg\_sel\_kl16z4\_t

Enumerator

*kSimAdcPretrgselA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

### 53.2.19.3.11 enum sim\_adc\_trg\_sel\_kl16z4\_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.

#### 53.2.19.3.12 enum sim\_uart\_rxsrc\_kl16z4\_t

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.  
*kSimUartRxsrcCmp0* CMP0.

#### 53.2.19.3.13 enum sim\_uart\_txsrc\_kl16z4\_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.

#### 53.2.19.3.14 enum sim\_lpsci\_rxsrc\_kl16z4\_t

Enumerator

*kSimLpsciRxsrcPin* LPSCIx\_RX Pin.  
*kSimLpsciRxsrcCmp0* CMP0.

#### 53.2.19.3.15 enum sim\_lpsci\_txsrc\_kl16z4\_t

Enumerator

*kSimLpsciTxsrcPin* LPSCIx\_TX Pin.

## SIM HAL driver

*kSimLpsciTxsrcTpm1* LPSCIx\_TX pin modulated with TPM1 channel 0 output.

*kSimLpsciTxsrcTpm2* LPSCIx\_TX pin modulated with TPM2 channel 0 output.

*kSimLpsciTxsrcReserved* Reserved.

### 53.2.19.3.16 enum sim\_tpm\_clk\_sel\_kl16z4\_t

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.

*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 53.2.19.3.17 enum sim\_tpm\_ch\_src\_kl16z4\_t

Enumerator

*kSimTpmChSrc0* TPMx\_CH0 signal.

*kSimTpmChSrc1* CMP0 output.

*kSimTpmChSrc2* Reserved.

*kSimTpmChSrc3* USB start of frame pulse.

### 53.2.19.3.18 enum sim\_clock\_gate\_name\_kl16z4\_t



## 53.2.20 KL17Z4 SIM HAL driver

### 53.2.20.1 Overview

The section describes the enumerations, macros and data structures for KL17Z4 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKL17Z4.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_cop\\_src\\_kl17z4\\_t](#) {  
  [kClockCopSrcLpoClk](#),  
  [kClockCopSrcMcgIrClk](#),  
  [kClockCopSrcOsc0erClk](#),  
  [kClockCopSrcBusClk](#) }  
*COP clock source selection.*
- enum [clock\\_er32k\\_src\\_kl17z4\\_t](#) {  
  [kClockEr32kSrcOsc0](#) = 0U,  
  [kClockEr32kSrcRtc](#) = 2U,  
  [kClockEr32kSrcLpo](#) = 3U }  
*SIM external reference clock source select (OSC32KSEL).*
- enum [clock\\_osc32kout\\_sel\\_kl17z4\\_t](#) {  
  [kClockOsc32koutNone](#) = 0U,  
  [kClockOsc32koutPte0](#) = 1U,  
  [kClockOsc32koutPte26](#) = 2U }  
*SIM external reference clock output pin select (OSC32KOUT).*
- enum [clock\\_lpuart\\_src\\_kl17z4\\_t](#) {  
  [kClockLpuartSrcNone](#),  
  [kClockLpuartSrcIrc48M](#),  
  [kClockLpuartSrcOsc0erClk](#),  
  [kClockLpuartSrcMcgIrClk](#) }  
*SIM LPUART clock source.*
- enum [clock\\_tpm\\_src\\_kl17z4\\_t](#) {  
  [kClockTpmSrcNone](#),  
  [kClockTpmSrcIrc48M](#),  
  [kClockTpmSrcOsc0erClk](#),  
  [kClockTpmSrcMcgIrClk](#) }  
*SIM TPM clock source.*

## SIM HAL driver

- enum `clock_flexio_src_kl17z4_t` {  
`kClockFlexioSrcNone`,  
`kClockFlexioSrcIrc48M`,  
`kClockFlexioSrcOsc0erClk`,  
`kClockFlexioSrcMcgIrClk` }  
*FLEXIO clock source.*
- enum `clock_lptmr_src_kl17z4_t` {  
`kClockLptmrSrcMcgIrClk`,  
`kClockLptmrSrcLpoClk`,  
`kClockLptmrSrcEr32kClk`,  
`kClockLptmrSrcOsc0erClk` }  
*LPTMR clock source select.*
- enum `clock_clkout_src_kl17z4_t` {  
`kClockClkoutSelFlashClk = 2U`,  
`kClockClkoutSelLpoClk = 3U`,  
`kClockClkoutSelMcgIrClk = 4U`,  
`kClockClkoutSelOsc0erClk = 6U`,  
`kClockClkoutSelIrc48M = 7U` }  
*SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_kl17z4_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrcOsc0erClk` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_kl17z4_t` {  
`kSimAdcPretrgselA`,  
`kSimAdcPretrgselB` }  
*SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kl17z4_t` {  
`kSimAdcTrgselExt = 0U`,  
`kSimAdcTrgSelComp0 = 1U`,  
`kSimAdcTrgSelPit0 = 4U`,  
`kSimAdcTrgSelPit1 = 5U`,  
`kSimAdcTrgSelTpm0 = 8U`,  
`kSimAdcTrgSelTpm1 = 9U`,  
`kSimAdcTrgSelTpm2 = 10U`,  
`kSimAdcTrgSelRtcAlarm = 12U`,  
`kSimAdcTrgSelRtcSec = 13U`,  
`kSimAdcTrgSelLptimer = 14U` }  
*SIM ADCx trigger select.*
- enum `sim_lpuart_rxsrc_kl17z4_t` {  
`kSimLpuartRxsrcPin`,  
`kSimLpuartRxsrcCmp0` }  
*LPUART receive data source.*
- enum `sim_lpuart_txsrc_kl17z4_t` {  
`kSimLpuartTxsrcPin`,  
`kSimLpuartTxsrcTpm1`,

- `kSimLpuartTxsrcTpm2` }  
*LPUART transmit data source.*
- enum `clock_sai_src_kl17z4_t` {  
`kClockSaiSrcSysClk` = 0U,  
`kClockSaiSrcOsc0erClk` = 1U,  
`kClockSaiSrcMcgIrClk` = 2U,  
`kClockSaiSrcIrc48M` = 3U }  
*SAI clock source.*
- enum `sim_tpm_clk_sel_kl17z4_t` {  
`kSimTpmClkSel0`,  
`kSimTpmClkSel1` }  
*SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_kl17z4_t` {  
`kSimTpmChSrc0`,  
`kSimTpmChSrc1`,  
`kSimTpmChSrc2`,  
`kSimTpmChSrc3` }  
*SIM Timer/PWM x channel y input capture source select.*
- enum `sim_clock_gate_name_kl17z4_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.20.2 Macro Definition Documentation

53.2.20.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.20.3 Enumeration Type Documentation

#### 53.2.20.3.1 enum `clock_cop_src_kl17z4_t`

Enumerator

*kClockCopSrcLpoClk* LPO clock 1K HZ.  
*kClockCopSrcMcgIrClk* MCG IRC Clock.  
*kClockCopSrcOsc0erClk* OSCER Clock.  
*kClockCopSrcBusClk* BUS clock.

#### 53.2.20.3.2 enum `clock_er32k_src_kl17z4_t`

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

## SIM HAL driver

### 53.2.20.3.3 enum clock\_osc32kout\_sel\_kl17z4\_t

Enumerator

*kClockOsc32koutNone* ERCLK32K is not output.  
*kClockOsc32koutPte0* ERCLK32K is output on PTE0.  
*kClockOsc32koutPte26* ERCLK32K is output on PTE26.

### 53.2.20.3.4 enum clock\_lpuart\_src\_kl17z4\_t

Enumerator

*kClockLpuartSrcNone* disabled  
*kClockLpuartSrcIrc48M* IRC48M.  
*kClockLpuartSrcOsc0erClk* OSCER clock.  
*kClockLpuartSrcMcgIrClk* MCGIR clock.

### 53.2.20.3.5 enum clock\_tpm\_src\_kl17z4\_t

Enumerator

*kClockTpmSrcNone* disabled  
*kClockTpmSrcIrc48M* IRC48M/MCGPCLK.  
*kClockTpmSrcOsc0erClk* OSCER clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.20.3.6 enum clock\_flexio\_src\_kl17z4\_t

Enumerator

*kClockFlexioSrcNone* Clock disabled.  
*kClockFlexioSrcIrc48M* MCGPCLK/IRC48M.  
*kClockFlexioSrcOsc0erClk* OSCERCLK.  
*kClockFlexioSrcMcgIrClk* MCGIRCLK.

### 53.2.20.3.7 enum clock\_lptmr\_src\_kl17z4\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.20.3.8 enum clock\_clkout\_src\_kl17z4\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelOsc0erClk* OSCER clock.  
*kClockClkoutSelIrc48M* IRC48M clock.

### 53.2.20.3.9 enum clock\_rtcout\_src\_kl17z4\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrcOsc0erClk* OSCER clock.

### 53.2.20.3.10 enum sim\_adc\_pretrg\_sel\_kl17z4\_t

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.20.3.11 enum sim\_adc\_trg\_sel\_kl17z4\_t

Enumerator

*kSimAdcTrgselExt* External trigger.  
*kSimAdcTrgSelComp0* CMP0 output.  
*kSimAdcTrgSelPit0* PIT trigger 0.  
*kSimAdcTrgSelPit1* PIT trigger 1.  
*kSimAdcTrgSelTpm0* TPM0 overflow.  
*kSimAdcTrgSelTpm1* TPM1 overflow.  
*kSimAdcTrgSelTpm2* TPM2 overflow.  
*kSimAdcTrgSelRtcAlarm* RTC alarm.  
*kSimAdcTrgSelRtcSec* RTC seconds.  
*kSimAdcTrgSelLptimer* Low-power timer trigger.

### 53.2.20.3.12 enum sim\_lpuart\_rxsrc\_kl17z4\_t

Enumerator

*kSimLpuartRxsrcPin* LPUARTx\_RX Pin.  
*kSimLpuartRxsrcCmp0* CMP0.

## SIM HAL driver

### 53.2.20.3.13 enum sim\_lpuart\_txsrc\_kl17z4\_t

Enumerator

- kSimLpuartTxsrcPin* UARTx\_TX Pin.
- kSimLpuartTxsrcTpm1* UARTx\_TX pin modulated with TPM1 channel 0 output.
- kSimLpuartTxsrcTpm2* UARTx\_TX pin modulated with TPM2 channel 0 output.

### 53.2.20.3.14 enum clock\_sai\_src\_kl17z4\_t

Enumerator

- kClockSaiSrcSysClk* SYSCLK.
- kClockSaiSrcOsc0erClk* OSC0ERCLK.
- kClockSaiSrcMcgIrClk* MCGIRCLK.
- kClockSaiSrcIrc48M* MCGPCLK/IRC48M.

### 53.2.20.3.15 enum sim\_tpm\_clk\_sel\_kl17z4\_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 53.2.20.3.16 enum sim\_tpm\_ch\_src\_kl17z4\_t

Enumerator

- kSimTpmChSrc0* Channel y input capture source uses 0.
- kSimTpmChSrc1* Channel y input capture source uses 1.
- kSimTpmChSrc2* Channel y input capture source uses 2.
- kSimTpmChSrc3* Channel y input capture source uses 3.

### 53.2.20.3.17 enum sim\_clock\_gate\_name\_kl17z4\_t

## 53.2.21 KL17Z644 SIM HAL driver

### 53.2.21.1 Overview

The section describes the enumerations, macros and data structures for KL17Z644 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKL17Z644.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_cop\\_src\\_kl17z644\\_t](#) {  
  [kClockCopSrcLpoClk](#),  
  [kClockCopSrcMcgIrClk](#),  
  [kClockCopSrcOsc0erClk](#),  
  [kClockCopSrcBusClk](#) }  
*COP clock source selection.*
- enum [clock\\_er32k\\_src\\_kl17z644\\_t](#) {  
  [kClockEr32kSrcOsc0](#) = 0U,  
  [kClockEr32kSrcRtc](#) = 2U,  
  [kClockEr32kSrcLpo](#) = 3U }  
*SIM external reference clock source select (OSC32KSEL).*
- enum [clock\\_osc32kout\\_sel\\_kl17z644\\_t](#) {  
  [kClockOsc32koutNone](#),  
  [kClockOsc32koutPte0](#) }  
*SIM external reference clock output pin select (OSC32KOUT).*
- enum [clock\\_lpuart\\_src\\_kl17z644\\_t](#) {  
  [kClockLpuartSrcNone](#),  
  [kClockLpuartSrcIrc48M](#),  
  [kClockLpuartSrcOsc0erClk](#),  
  [kClockLpuartSrcMcgIrClk](#) }  
*SIM LPUART clock source.*
- enum [clock\\_tpm\\_src\\_kl17z644\\_t](#) {  
  [kClockTpmSrcNone](#),  
  [kClockTpmSrcIrc48M](#),  
  [kClockTpmSrcOsc0erClk](#),  
  [kClockTpmSrcMcgIrClk](#) }  
*SIM TPM clock source.*

## SIM HAL driver

- enum `clock_usbfs_src_kl17z644_t` {  
`kClockUsbfsSrcExt`,  
`kClockUsbfsSrcIrc48M` }  
*SIM USB FS clock source.*
- enum `clock_flexio_src_kl17z644_t` {  
`kClockFlexioSrcNone`,  
`kClockFlexioSrcIrc48M`,  
`kClockFlexioSrcOsc0erClk`,  
`kClockFlexioSrcMcgIrClk` }  
*FLEXIO clock source.*
- enum `clock_lptmr_src_kl17z644_t` {  
`kClockLptmrSrcMcgIrClk`,  
`kClockLptmrSrcLpoClk`,  
`kClockLptmrSrcEr32kClk`,  
`kClockLptmrSrcOsc0erClk` }  
*LPTMR clock source select.*
- enum `clock_clkout_src_kl17z644_t` {  
`kClockClkoutSelFlashClk` = 2U,  
`kClockClkoutSelLpoClk` = 3U,  
`kClockClkoutSelMcgIrClk` = 4U,  
`kClockClkoutSelOsc0erClk` = 6U,  
`kClockClkoutSelIrc48M` = 7U }  
*SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_kl17z644_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrcOsc0erClk` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_kl17z644_t` {  
`kSimAdcPretrgselA`,  
`kSimAdcPretrgselB` }  
*SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kl17z644_t` {  
`kSimAdcTrgselExt` = 0U,  
`kSimAdcTrgSelComp0` = 1U,  
`kSimAdcTrgSelPit0` = 4U,  
`kSimAdcTrgSelPit1` = 5U,  
`kSimAdcTrgSelTpm0` = 8U,  
`kSimAdcTrgSelTpm1` = 9U,  
`kSimAdcTrgSelTpm2` = 10U,  
`kSimAdcTrgSelRtcAlarm` = 12U,  
`kSimAdcTrgSelRtcSec` = 13U,  
`kSimAdcTrgSelLptimer` = 14U }  
*SIM ADCx trigger select.*
- enum `sim_lpuart_rxs_src_kl17z644_t` {  
`kSimLpuartRxsSrcPin`,  
`kSimLpuartRxsSrcCmp0` }



- LPUART receive data source.*

  - enum `sim_lpuart_txsrc_kl17z644_t` {  
`kSimLpuartTxsrcPin`,  
`kSimLpuartTxsrcTpm1`,  
`kSimLpuartTxsrcTpm2` }
- LPUART transmit data source.*

  - enum `sim_tpm_clk_sel_kl17z644_t` {  
`kSimTpmClkSel0`,  
`kSimTpmClkSel1` }
- SIM Timer/PWM external clock select.*

  - enum `sim_tpm_ch_src_kl17z644_t` {  
`kSimTpmChSrc0`,  
`kSimTpmChSrc1`,  
`kSimTpmChSrc2`,  
`kSimTpmChSrc3` }
- SIM Timer/PWM x channel y input capture source select.*

  - enum `sim_clock_gate_name_kl17z644_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

## 53.2.21.2 Macro Definition Documentation

53.2.21.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

## 53.2.21.3 Enumeration Type Documentation

### 53.2.21.3.1 enum `clock_cop_src_kl17z644_t`

Enumerator

*kClockCopSrcLpoClk* LPO clock 1K HZ.  
*kClockCopSrcMcgIrClk* MCG IRC Clock.  
*kClockCopSrcOsc0erClk* OSCER Clock.  
*kClockCopSrcBusClk* BUS clock.

### 53.2.21.3.2 enum `clock_er32k_src_kl17z644_t`

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

## SIM HAL driver

### 53.2.21.3.3 enum clock\_osc32kout\_sel\_kl17z644\_t

Enumerator

*kClockOsc32koutNone* ERCLK32K is not output.  
*kClockOsc32koutPte0* ERCLK32K output on PTE0.

### 53.2.21.3.4 enum clock\_lpuart\_src\_kl17z644\_t

Enumerator

*kClockLpuartSrcNone* disabled  
*kClockLpuartSrcIrc48M* IRC48M.  
*kClockLpuartSrcOsc0erClk* OSCER clock.  
*kClockLpuartSrcMcgIrClk* MCGIR clock.

### 53.2.21.3.5 enum clock\_tpm\_src\_kl17z644\_t

Enumerator

*kClockTpmSrcNone* disabled  
*kClockTpmSrcIrc48M* IRC48M/MCGPCLK.  
*kClockTpmSrcOsc0erClk* OSCER clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.21.3.6 enum clock\_usbfs\_src\_kl17z644\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)  
*kClockUsbfsSrcIrc48M* IRC48/MCGPCLK.

### 53.2.21.3.7 enum clock\_flexio\_src\_kl17z644\_t

Enumerator

*kClockFlexioSrcNone* Clock disabled.  
*kClockFlexioSrcIrc48M* MCGPCLK/IRC48M.  
*kClockFlexioSrcOsc0erClk* OSCERCLK.  
*kClockFlexioSrcMcgIrClk* MCGIRCLK.

### 53.2.21.3.8 enum clock\_lptmr\_src\_kl17z644\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.21.3.9 enum clock\_clkout\_src\_kl17z644\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelOsc0erClk* OSCER clock.  
*kClockClkoutSelIrc48M* IRC48M clock.

### 53.2.21.3.10 enum clock\_rtcout\_src\_kl17z644\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrcOsc0erClk* OSCER clock.

### 53.2.21.3.11 enum sim\_adc\_pretrg\_sel\_kl17z644\_t

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.21.3.12 enum sim\_adc\_trg\_sel\_kl17z644\_t

Enumerator

*kSimAdcTrgSelExt* External trigger.  
*kSimAdcTrgSelComp0* CMP0 output.  
*kSimAdcTrgSelPit0* PIT trigger 0.  
*kSimAdcTrgSelPit1* PIT trigger 1.  
*kSimAdcTrgSelTpm0* TPM0 overflow.  
*kSimAdcTrgSelTpm1* TPM1 overflow.

## SIM HAL driver

*kSimAdcTrgSelTpm2* TPM2 overflow.  
*kSimAdcTrgSelRtcAlarm* RTC alarm.  
*kSimAdcTrgSelRtcSec* RTC seconds.  
*kSimAdcTrgSelLptimer* Low-power timer trigger.

### 53.2.21.3.13 enum sim\_lpuart\_rxsrc\_kl17z644\_t

Enumerator

*kSimLpuartRxsrcPin* LPUARTx\_RX Pin.  
*kSimLpuartRxsrcCmp0* CMP0.

### 53.2.21.3.14 enum sim\_lpuart\_txsrc\_kl17z644\_t

Enumerator

*kSimLpuartTxsrcPin* UARTx\_TX Pin.  
*kSimLpuartTxsrcTpm1* UARTx\_TX pin modulated with TPM1 channel 0 output.  
*kSimLpuartTxsrcTpm2* UARTx\_TX pin modulated with TPM2 channel 0 output.

### 53.2.21.3.15 enum sim\_tpm\_clk\_sel\_kl17z644\_t

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.  
*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 53.2.21.3.16 enum sim\_tpm\_ch\_src\_kl17z644\_t

Enumerator

*kSimTpmChSrc0* Channel y input capture source uses 0.  
*kSimTpmChSrc1* Channel y input capture source uses 1.  
*kSimTpmChSrc2* Channel y input capture source uses 2.  
*kSimTpmChSrc3* Channel y input capture source uses 3.

### 53.2.21.3.17 enum sim\_clock\_gate\_name\_kl17z644\_t

## 53.2.22 K21FA12 SIM HAL driver

### 53.2.22.1 Overview

The section describes the enumerations, macros and data structures for K21FA12 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MK21FA12.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_k21fa12\\_t](#) {  
  [kClockWdogSrcLpoClk](#),  
  [kClockWdogSrcAltClk](#) }  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_k21fa12\\_t](#) {  
  [kClockTraceSrcMcgoutClk](#),  
  [kClockTraceSrcCoreClk](#) }  
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_k21fa12\\_t](#) {  
  [kClockPortFilterSrcBusClk](#),  
  [kClockPortFilterSrcLpoClk](#) }  
*PORTx digital input filter clock source select.*
- enum [clock\\_lptmr\\_src\\_k21fa12\\_t](#) {  
  [kClockLptmrSrcMcgIrClk](#),  
  [kClockLptmrSrcLpoClk](#),  
  [kClockLptmrSrcEr32kClk](#),  
  [kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_usbfs\\_src\\_k21fa12\\_t](#) {  
  [kClockUsbfsSrcExt](#),  
  [kClockUsbfsSrcPllFllSel](#) }  
*SIM USB FS clock source.*
- enum [clock\\_flexcan\\_src\\_k21fa12\\_t](#) {  
  [kClockFlexcanSrcOsc0erClk](#),  
  [kClockFlexcanSrcBusClk](#) }  
*FLEXCAN clock source select.*
- enum [clock\\_sdhc\\_src\\_k21fa12\\_t](#) {

## SIM HAL driver

- kClockSdhcSrcCoreSysClk,
  - kClockSdhcSrcPllFllSel,
  - kClockSdhcSrcOsc0erClk,
  - kClockSdhcSrcExt }

*SDHC clock source.*
- enum clock\_sai\_src\_k21fa12\_t {
  - kClockSaiSrcSysClk = 0U,
  - kClockSaiSrcOsc0erClk = 1U,
  - kClockSaiSrcPllClk = 3U }

*SAI clock source.*
- enum clock\_pllfl\_sel\_k21fa12\_t {
  - kClockPllFllSelFll = 0U,
  - kClockPllFllSelPll = 1U,
  - kClockPllFllSelIrc48M = 3U }

*SIM PLLFLLSEL clock source select.*
- enum clock\_er32k\_src\_k21fa12\_t {
  - kClockEr32kSrcOsc0 = 0U,
  - kClockEr32kSrcRtc = 2U,
  - kClockEr32kSrcLpo = 3U }

*SIM external reference clock source select (OSC32KSEL).*
- enum clock\_clkout\_src\_k21fa12\_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\_k21fa12\_t {
  - kClockRtcoutSrc1Hz,
  - kClockRtcoutSrc32kHz }

*SIM RTCCLKOUTSEL clock source select.*
- enum sim\_usbsstby\_mode\_k21fa12\_t {
  - kSimUsbsstbyNoRegulator,
  - kSimUsbsstbyWithRegulator }

*SIM USB voltage regulator in standby mode setting during stop modes.*
- enum sim\_usbvstby\_mode\_k21fa12\_t {
  - kSimUsbvstbyNoRegulator,
  - kSimUsbvstbyWithRegulator }

*SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.*
- enum sim\_adc\_pretrg\_sel\_k21fa12\_t {
  - kSimAdcPretrgselA,
  - kSimAdcPretrgselB }

*SIM ADCx pre-trigger select.*
- enum sim\_adc\_trg\_sel\_k21fa12\_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_k21fa12_t` {  
  - kSimUartRxsrcPin,
  - kSimUartRxsrcCmp0,
  - kSimUartRxsrcCmp1 }*SIM UART receive data source select.*
- enum `sim_uart_txsrc_k21fa12_t` {  
  - kSimUartTxsrcPin,
  - kSimUartTxsrcFtm1,
  - kSimUartTxsrcFtm2 }*SIM UART transmit data source select.*
- enum `sim_ftm_trg_src_k21fa12_t` {  
  - kSimFtmTrgSrc0,
  - kSimFtmTrgSrc1 }*SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_k21fa12_t` {  
  - kSimFtmClkSel0,
  - kSimFtmClkSel1 }*SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_k21fa12_t` {  
  - kSimFtmChSrc0,
  - kSimFtmChSrc1,
  - kSimFtmChSrc2,
  - kSimFtmChSrc3 }*SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftmflt_sel_k21fa12_t` {  
  - kSimFtmFltSel0,
  - kSimFtmFltSel1 }*SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_k21fa12_t` {  
  - kSimTpmClkSel0,

## SIM HAL driver

- `kSimTpmClkSel1` }  
*SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_k21fa12_t` {  
`kSimTpmChSrc0`,  
`kSimTpmChSrc1`,  
`kSimTpmChSrc2`,  
`kSimTpmChSrc3` }  
*SIM Timer/PWM x channel y input capture source select.*
- enum `sim_cmtuartpad_strenght_k21fa12_t` {  
`kSimCmtuartSinglePad`,  
`kSimCmtuartDualPad` }  
*SIM CMT/UART pad drive strength.*
- enum `sim_ptd7pad_strenght_k21fa12_t` {  
`kSimPtd7padSinglePad`,  
`kSimPtd7padDualPad` }  
*SIM PTD7 pad drive strength.*
- enum `sim_flexbus_security_level_k21fa12_t` {  
`kSimFbslLevel0`,  
`kSimFbslLevel1`,  
`kSimFbslLevel2`,  
`kSimFbslLevel3` }  
*SIM FlexBus security level.*
- enum `sim_clock_gate_name_k21fa12_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.22.2 Macro Definition Documentation

53.2.22.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.22.3 Enumeration Type Documentation

53.2.22.3.1 enum `clock_wdog_src_k21fa12_t`

Enumerator

*kClockWdogSrcLpoClk* LPO.

*kClockWdogSrcAltClk* Alternative clock, for MK21FA12 it is Bus clock.

53.2.22.3.2 enum `clock_trace_src_k21fa12_t`

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock



### 53.2.22.3.3 enum clock\_port\_filter\_src\_k21fa12\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.

*kClockPortFilterSrcLpoClk* LPO.

### 53.2.22.3.4 enum clock\_lptmr\_src\_k21fa12\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.

*kClockLptmrSrcLpoClk* LPO clock.

*kClockLptmrSrcEr32kClk* ERCLK32K clock.

*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.22.3.5 enum clock\_usbfs\_src\_k21fa12\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)

*kClockUsbfsSrcPIFUSel* Clock divider USB FS clock.

### 53.2.22.3.6 enum clock\_flexcan\_src\_k21fa12\_t

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.

*kClockFlexcanSrcBusClk* Bus clock.

### 53.2.22.3.7 enum clock\_sdhc\_src\_k21fa12\_t

Enumerator

*kClockSdhcSrcCoreSysClk* Core/system clock.

*kClockSdhcSrcPIFUSel* clock as selected by SOPT2[PLLFLSEL].

*kClockSdhcSrcOsc0erClk* OSCERCLK clock.

*kClockSdhcSrcExt* External bypass clock (SDHC0\_CLKIN)

## SIM HAL driver

### 53.2.22.3.8 enum clock\_sai\_src\_k21fa12\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.22.3.9 enum clock\_pllfl\_sel\_k21fa12\_t

Enumerator

*kClockPllFlSelFll* Fll clock.  
*kClockPllFlSelPll* Pll0 clock.  
*kClockPllFlSelIrc48M* IRC48MCLK.

### 53.2.22.3.10 enum clock\_er32k\_src\_k21fa12\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.22.3.11 enum clock\_clkout\_src\_k21fa12\_t

Enumerator

*kClockClkoutSelFlexbusClk* Flexbus clock.  
*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCGIRCLK.  
*kClockClkoutSelRtc* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSC0ERCLK.  
*kClockClkoutSelIrc48M* IRC48MCLK.

### 53.2.22.3.12 enum clock\_rtcout\_src\_k21fa12\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

### 53.2.22.3.13 enum sim\_usbsstby\_mode\_k21fa12\_t

Enumerator

*kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes

*kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

### 53.2.22.3.14 enum sim\_usbvstby\_mode\_k21fa12\_t

Enumerator

*kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes

*kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes

### 53.2.22.3.15 enum sim\_adc\_pretrg\_sel\_k21fa12\_t

Enumerator

*kSimAdcPretrgselA* Pre-trigger A selected for ADCx.

*kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

### 53.2.22.3.16 enum sim\_adc\_trg\_sel\_k21fa12\_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.

## SIM HAL driver

### 53.2.22.3.17 enum sim\_uart\_rxsrc\_k21fa12\_t

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.

*kSimUartRxsrcCmp0* CMP0.

*kSimUartRxsrcCmp1* CMP1.

### 53.2.22.3.18 enum sim\_uart\_txsrc\_k21fa12\_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.

### 53.2.22.3.19 enum sim\_ftm\_trg\_src\_k21fa12\_t

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.

*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

### 53.2.22.3.20 enum sim\_ftm\_clk\_sel\_k21fa12\_t

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.

*kSimFtmClkSel1* FTM CLKIN1 pin.

### 53.2.22.3.21 enum sim\_ftm\_ch\_src\_k21fa12\_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.

### 53.2.22.3.22 enum sim\_ftmflt\_sel\_k21fa12\_t

Enumerator

*kSimFtmFltSel0* FlexTimer x fault y select 0.

*kSimFtmFltSel1* FlexTimer x fault y select 1.

**53.2.22.3.23 enum sim\_tpm\_clk\_sel\_k21fa12\_t**

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

**53.2.22.3.24 enum sim\_tpm\_ch\_src\_k21fa12\_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.

**53.2.22.3.25 enum sim\_cmtuartpad\_streng\_h\_k21fa12\_t**

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.

**53.2.22.3.26 enum sim\_ptd7pad\_streng\_h\_k21fa12\_t**

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

**53.2.22.3.27 enum sim\_flexbus\_security\_level\_k21fa12\_t**

Enumerator

- kSimFbslLevel0* FlexBus security level 0.
- kSimFbslLevel1* FlexBus security level 1.
- kSimFbslLevel2* FlexBus security level 2.
- kSimFbslLevel3* FlexBus security level 3.

**53.2.22.3.28 enum sim\_clock\_gate\_name\_k21fa12\_t**

## SIM HAL driver

### 53.2.23 K24F12 SIM HAL driver

#### 53.2.23.1 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](#) {

- kClockSdheSrcCoreSysClk,
- kClockSdheSrcPllFllSel,
- kClockSdheSrcOsc0erClk,
- kClockSdheSrcExt }
- SDHC clock source.*
- enum clock\_sai\_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 {
            - kSimUsbsstbyNoRegulator,
            - kSimUsbsstbyWithRegulator }
            - SIM USB voltage regulator in standby mode setting during stop modes.*
            - enum sim\_usbvstby\_mode\_k24f12\_t {
              - kSimUsbvstbyNoRegulator,
              - kSimUsbvstbyWithRegulator }
              - 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 UART receive data source select.*
- enum sim\_uart\_txsrc\_k24f12\_t {
  - kSimUartTxsrcPin,
  - kSimUartTxsrcFtm1,
  - kSimUartTxsrcFtm2 }
  - SIM UART 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\_ftmflt\_sel\_k24f12\_t {
  - kSimFtmFltSel0,
  - kSimFtmFltSel1 }
  - 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_streng_h_k24f12_t` {  
`kSimCmtuartSinglePad`,  
`kSimCmtuartDualPad` }  
*SIM CMT/UART pad drive strength.*
- enum `sim_ptd7pad_streng_h_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.*

### 53.2.23.2 Macro Definition Documentation

53.2.23.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.23.3 Enumeration Type Documentation

53.2.23.3.1 enum `clock_wdog_src_k24f12_t`

Enumerator

*kClockWdogSrcLpoClk* LPO.

*kClockWdogSrcAltClk* Alternative clock, for this SOC it is Bus clock.

53.2.23.3.2 enum `clock_trace_src_k24f12_t`

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

## SIM HAL driver

### 53.2.23.3.3 enum clock\_port\_filter\_src\_k24f12\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.  
*kClockPortFilterSrcLpoClk* LPO.

### 53.2.23.3.4 enum clock\_lptmr\_src\_k24f12\_t

Enumerator

*kClockLptmrSrcMcgrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.23.3.5 enum clock\_usbfs\_src\_k24f12\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)  
*kClockUsbfsSrcPIFUSel* Clock divider USB FS clock.

### 53.2.23.3.6 enum clock\_flexcan\_src\_k24f12\_t

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.

### 53.2.23.3.7 enum clock\_sdhc\_src\_k24f12\_t

Enumerator

*kClockSdhcSrcCoreSysClk* Core/system clock.  
*kClockSdhcSrcPIFUSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockSdhcSrcOsc0erClk* OSCERCLK clock.  
*kClockSdhcSrcExt* External bypass clock (SDHC0\_CLKIN)

### 53.2.23.3.8 enum clock\_sai\_src\_k24f12\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.23.3.9 enum clock\_pllfl\_sel\_k24f12\_t

Enumerator

*kClockPllFlSelFll* Fll clock.  
*kClockPllFlSelPll* Pll0 clock.  
*kClockPllFlSelIrc48M* IRC48MCLK.

### 53.2.23.3.10 enum clock\_er32k\_src\_k24f12\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.23.3.11 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.

### 53.2.23.3.12 enum clock\_rtcout\_src\_k24f12\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

## SIM HAL driver

### 53.2.23.3.13 enum sim\_usbsstby\_mode\_k24f12\_t

Enumerator

*kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes

*kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

### 53.2.23.3.14 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

### 53.2.23.3.15 enum sim\_adc\_pretrg\_sel\_k24f12\_t

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.

*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.23.3.16 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.

**53.2.23.3.17 enum sim\_uart\_rxsrc\_k24f12\_t**

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.*kSimUartRxsrcCmp0* CMP0.*kSimUartRxsrcCmp1* CMP1.**53.2.23.3.18 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.**53.2.23.3.19 enum sim\_ftm\_trg\_src\_k24f12\_t**

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.**53.2.23.3.20 enum sim\_ftm\_clk\_sel\_k24f12\_t**

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.*kSimFtmClkSel1* FTM CLKIN1 pin.**53.2.23.3.21 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.**53.2.23.3.22 enum sim\_ftmflt\_sel\_k24f12\_t**

Enumerator

*kSimFtmFltSel0* FlexTimer x fault y select 0.*kSimFtmFltSel1* FlexTimer x fault y select 1.

## SIM HAL driver

### 53.2.23.3.23 enum sim\_tpm\_clk\_sel\_k24f12\_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 53.2.23.3.24 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.

### 53.2.23.3.25 enum sim\_cmtuartpad\_strength\_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.

### 53.2.23.3.26 enum sim\_ptd7pad\_strength\_k24f12\_t

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

### 53.2.23.3.27 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.

### 53.2.23.3.28 enum sim\_clock\_gate\_name\_k24f12\_t

## 53.2.24 KL26Z4 SIM HAL driver

### 53.2.24.1 Overview

The section describes the enumerations, macros and data structures for KL26Z4 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKL26Z4.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_cop\\_src\\_kl26z4\\_t](#) {  
  [kClockCopSrcLpoClk](#),  
  [kClockCopSrcAltClk](#) }  
*COP clock source select.*
- enum [clock\\_tpm\\_src\\_kl26z4\\_t](#) {  
  [kClockTpmSrcNone](#),  
  [kClockTpmSrcPllFllSel](#),  
  [kClockTpmSrcOsc0erClk](#),  
  [kClockTpmSrcMcgIrClk](#) }  
*TPM clock source select.*
- enum [clock\\_lptmr\\_src\\_kl26z4\\_t](#) {  
  [kClockLptmrSrcMcgIrClk](#),  
  [kClockLptmrSrcLpoClk](#),  
  [kClockLptmrSrcEr32kClk](#),  
  [kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_lpsci\\_src\\_kl26z4\\_t](#) {  
  [kClockLpsciSrcNone](#),  
  [kClockLpsciSrcPllFllSel](#),  
  [kClockLpsciSrcOsc0erClk](#),  
  [kClockLpsciSrcMcgIrClk](#) }  
*UART0 clock source select.*
- enum [clock\\_sai\\_src\\_kl26z4\\_t](#) {  
  [kClockSaiSrcSysClk](#) = 0U,  
  [kClockSaiSrcOsc0erClk](#) = 1U,  
  [kClockSaiSrcPllClk](#) = 3U }  
*SAI clock source.*

## SIM HAL driver

- enum `clock_pllfl1_sel_kl26z4_t` {  
`kClockPllFl1SelFl1`,  
`kClockPllFl1SelPll` }  
*SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_kl26z4_t` {  
`kClockEr32kSrcOsc0` = 0U,  
`kClockEr32kSrcReserved` = 1U,  
`kClockEr32kSrcRtc` = 2U,  
`kClockEr32kSrcLpo` = 3U }  
*SIM external reference clock source select (OSC32KSEL)*
- enum `clock_clkout_src_kl26z4_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_kl26z4_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrc32kHz` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_kl26z4_t` {  
`kSimAdcPretrgselA`,  
`kSimAdcPretrgselB` }  
*SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kl26z4_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_rxsrc_kl26z4_t` {  
`kSimUartRxsrcPin`,  
`kSimUartRxsrcCmp0` }  
*SIM UART receive data source select.*
- enum `sim_uart_txsrc_kl26z4_t` {  
`kSimUartTxsrcPin`,  
`kSimUartTxsrcTpm1`,  
`kSimUartTxsrcTpm2`,  
`kSimUartTxsrcReserved` }  
*SIM UART transmit data source select.*
- enum `sim_lpsci_rxsrc_kl26z4_t` {  
`kSimLpsciRxsrcPin`,  
`kSimLpsciRxsrcCmp0` }  
*SIM LPSCI receive data source select.*
- enum `sim_lpsci_txsrc_kl26z4_t` {  
`kSimLpsciTxsrcPin`,  
`kSimLpsciTxsrcTpm1`,  
`kSimLpsciTxsrcTpm2`,  
`kSimLpsciTxsrcReserved` }  
*SIM LPSCI transmit data source select.*
- enum `sim_tpm_clk_sel_kl26z4_t` {  
`kSimTpmClkSel0`,  
`kSimTpmClkSel1` }  
*SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_kl26z4_t` {  
`kSimTpmChSrc0`,  
`kSimTpmChSrc1`,  
`kSimTpmChSrc2`,  
`kSimTpmChSrc3` }  
*SIM Timer/PWM x channel y input capture source select.*
- enum `sim_clock_gate_name_kl26z4_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

## 53.2.24.2 Macro Definition Documentation

53.2.24.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

## 53.2.24.3 Enumeration Type Documentation

53.2.24.3.1 enum `clock_cop_src_kl26z4_t`

Enumerator

*kClockCopSrcLpoClk* LPO.

*kClockCopSrcAltClk* Alternative clock, for KL26Z4 it is Bus clock.

## SIM HAL driver

### 53.2.24.3.2 enum clock\_tpm\_src\_kl26z4\_t

Enumerator

*kClockTpmSrcNone* clock disabled  
*kClockTpmSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockTpmSrcOsc0erClk* OSCERCLK clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.24.3.3 enum clock\_lptmr\_src\_kl26z4\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.24.3.4 enum clock\_lpsci\_src\_kl26z4\_t

Enumerator

*kClockLpsciSrcNone* clock disabled  
*kClockLpsciSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockLpsciSrcOsc0erClk* OSCERCLK clock.  
*kClockLpsciSrcMcgIrClk* MCGIR clock.

### 53.2.24.3.5 enum clock\_sai\_src\_kl26z4\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.24.3.6 enum clock\_pllfl\_sel\_kl26z4\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.

**53.2.24.3.7 enum clock\_er32k\_src\_kl26z4\_t**

Enumerator

*kClockEr32kSrcOsc0* OSC 32k clock.  
*kClockEr32kSrcReserved* Reserved.  
*kClockEr32kSrcRtc* RTC 32k clock.  
*kClockEr32kSrcLpo* LPO clock.

**53.2.24.3.8 enum clock\_clkout\_src\_kl26z4\_t**

Enumerator

*kClockClkoutReserved* Reserved.  
*kClockClkoutReserved1* Reserved.  
*kClockClkoutBusClk* Bus clock.  
*kClockClkoutLpoClk* LPO clock.  
*kClockClkoutMcgIrClk* MCG ir clock.  
*kClockClkoutReserved2* Reserved.  
*kClockClkoutOsc0erClk* OSC0ER clock.  
*kClockClkoutReserved3* Reserved.

**53.2.24.3.9 enum clock\_rtcout\_src\_kl26z4\_t**

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32KHz clock

**53.2.24.3.10 enum sim\_adc\_pretrg\_sel\_kl26z4\_t**

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

**53.2.24.3.11 enum sim\_adc\_trg\_sel\_kl26z4\_t**

Enumerator

*kSimAdcTrgSelExt* External trigger.  
*kSimAdcTrgSelComp0* CMP0 output.  
*kSimAdcTrgSelReserved* Reserved.

## SIM HAL driver

*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.

### 53.2.24.3.12 enum sim\_uart\_rxsrc\_kl26z4\_t

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.  
*kSimUartRxsrcCmp0* CMP0.

### 53.2.24.3.13 enum sim\_uart\_txsrc\_kl26z4\_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.

### 53.2.24.3.14 enum sim\_lpsci\_rxsrc\_kl26z4\_t

Enumerator

*kSimLpsciRxsrcPin* LPSCIx\_RX Pin.  
*kSimLpsciRxsrcCmp0* CMP0.

### 53.2.24.3.15 enum sim\_lpsci\_txsrc\_kl26z4\_t

Enumerator

*kSimLpsciTxsrcPin* LPSCIx\_TX Pin.

*kSimLpsciTxsrcTpm1* LPSCIx\_TX pin modulated with TPM1 channel 0 output.  
*kSimLpsciTxsrcTpm2* LPSCIx\_TX pin modulated with TPM2 channel 0 output.  
*kSimLpsciTxsrcReserved* Reserved.

#### 53.2.24.3.16 enum sim\_tpm\_clk\_sel\_kl26z4\_t

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.  
*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

#### 53.2.24.3.17 enum sim\_tpm\_ch\_src\_kl26z4\_t

Enumerator

*kSimTpmChSrc0* TPMx\_CH0 signal.  
*kSimTpmChSrc1* CMP0 output.  
*kSimTpmChSrc2* Reserved.  
*kSimTpmChSrc3* USB start of frame pulse.

#### 53.2.24.3.18 enum sim\_clock\_gate\_name\_kl26z4\_t

## SIM HAL driver

### 53.2.25 KL27Z4 SIM HAL driver

#### 53.2.25.1 Overview

The section describes the enumerations, macros and data structures for KL27Z4 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKL27Z4.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_cop\\_src\\_kl27z4\\_t](#) {  
[kClockCopSrcLpoClk](#),  
[kClockCopSrcMcgIrClk](#),  
[kClockCopSrcOsc0erClk](#),  
[kClockCopSrcBusClk](#) }  
*COP clock source selection.*
- enum [clock\\_er32k\\_src\\_kl27z4\\_t](#) {  
[kClockEr32kSrcOsc0](#) = 0U,  
[kClockEr32kSrcRtc](#) = 2U,  
[kClockEr32kSrcLpo](#) = 3U }  
*SIM external reference clock source select (OSC32KSEL).*
- enum [clock\\_osc32kout\\_sel\\_kl27z4\\_t](#) {  
[kClockOsc32koutNone](#) = 0U,  
[kClockOsc32koutPte0](#) = 1U,  
[kClockOsc32koutPte26](#) = 2U }  
*SIM external reference clock output pin select (OSC32KOUT).*
- enum [clock\\_lpuart\\_src\\_kl27z4\\_t](#) {  
[kClockLpuartSrcNone](#),  
[kClockLpuartSrcIrc48M](#),  
[kClockLpuartSrcOsc0erClk](#),  
[kClockLpuartSrcMcgIrClk](#) }  
*SIM LPUART clock source.*
- enum [clock\\_tpm\\_src\\_kl27z4\\_t](#) {  
[kClockTpmSrcNone](#),  
[kClockTpmSrcIrc48M](#),  
[kClockTpmSrcOsc0erClk](#),  
[kClockTpmSrcMcgIrClk](#) }  
*SIM TPM clock source.*

- enum `clock_flexio_src_kl27z4_t` {  
`kClockFlexioSrcNone`,  
`kClockFlexioSrcIrc48M`,  
`kClockFlexioSrcOsc0erClk`,  
`kClockFlexioSrcMcgIrClk` }  
*FLEXIO clock source.*
- enum `clock_lptmr_src_kl27z4_t` {  
`kClockLptmrSrcMcgIrClk`,  
`kClockLptmrSrcLpoClk`,  
`kClockLptmrSrcEr32kClk`,  
`kClockLptmrSrcOsc0erClk` }  
*LPTMR clock source select.*
- enum `clock_clkout_src_kl27z4_t` {  
`kClockClkoutSelFlashClk` = 2U,  
`kClockClkoutSelLpoClk` = 3U,  
`kClockClkoutSelMcgIrClk` = 4U,  
`kClockClkoutSelOsc0erClk` = 6U,  
`kClockClkoutSelIrc48M` = 7U }  
*SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_kl27z4_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrcOsc0erClk` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_kl27z4_t` {  
`kSimAdcPretrgselA`,  
`kSimAdcPretrgselB` }  
*SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kl27z4_t` {  
`kSimAdcTrgselExt` = 0U,  
`kSimAdcTrgSelComp0` = 1U,  
`kSimAdcTrgSelPit0` = 4U,  
`kSimAdcTrgSelPit1` = 5U,  
`kSimAdcTrgSelTpm0` = 8U,  
`kSimAdcTrgSelTpm1` = 9U,  
`kSimAdcTrgSelTpm2` = 10U,  
`kSimAdcTrgSelRtcAlarm` = 12U,  
`kSimAdcTrgSelRtcSec` = 13U,  
`kSimAdcTrgSelLptimer` = 14U }  
*SIM ADCx trigger select.*
- enum `sim_lpuart_rxsrc_kl27z4_t` {  
`kSimLpuartRxsrcPin`,  
`kSimLpuartRxsrcCmp0` }  
*LPUART receive data source.*
- enum `sim_lpuart_txsrc_kl27z4_t` {  
`kSimLpuartTxsrcPin`,  
`kSimLpuartTxsrcTpm1`,

## SIM HAL driver

- `kSimLpuartTxsrcTpm2` }  
*LPUART transmit data source.*
- enum `clock_sai_src_kl27z4_t` {  
`kClockSaiSrcSysClk` = 0U,  
`kClockSaiSrcOsc0erClk` = 1U,  
`kClockSaiSrcMcgIrClk` = 2U,  
`kClockSaiSrcIrc48M` = 3U }
- SAI clock source.*
- enum `sim_tpm_clk_sel_kl27z4_t` {  
`kSimTpmClkSel0`,  
`kSimTpmClkSel1` }
- SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_kl27z4_t` {  
`kSimTpmChSrc0`,  
`kSimTpmChSrc1`,  
`kSimTpmChSrc2`,  
`kSimTpmChSrc3` }
- SIM Timer/PWM x channel y input capture source select.*
- enum `sim_clock_gate_name_kl27z4_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.25.2 Macro Definition Documentation

53.2.25.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.25.3 Enumeration Type Documentation

53.2.25.3.1 enum `clock_cop_src_kl27z4_t`

Enumerator

`kClockCopSrcLpoClk` LPO clock 1K HZ.  
`kClockCopSrcMcgIrClk` MCG IRC Clock.  
`kClockCopSrcOsc0erClk` OSCER Clock.  
`kClockCopSrcBusClk` BUS clock.

53.2.25.3.2 enum `clock_er32k_src_kl27z4_t`

Enumerator

`kClockEr32kSrcOsc0` OSC0 clock (OSC032KCLK).  
`kClockEr32kSrcRtc` RTC 32k clock .  
`kClockEr32kSrcLpo` LPO clock.



### 53.2.25.3.3 enum clock\_osc32kout\_sel\_kl27z4\_t

Enumerator

*kClockOsc32koutNone* ERCLK32K is not output.  
*kClockOsc32koutPte0* ERCLK32K is output on PTE0.  
*kClockOsc32koutPte26* ERCLK32K is output on PTE26.

### 53.2.25.3.4 enum clock\_lpuart\_src\_kl27z4\_t

Enumerator

*kClockLpuartSrcNone* disabled  
*kClockLpuartSrcIrc48M* IRC48M.  
*kClockLpuartSrcOsc0erClk* OSCER clock.  
*kClockLpuartSrcMcgIrClk* MCGIR clock.

### 53.2.25.3.5 enum clock\_tpm\_src\_kl27z4\_t

Enumerator

*kClockTpmSrcNone* disabled  
*kClockTpmSrcIrc48M* IRC48M/MCGPCLK.  
*kClockTpmSrcOsc0erClk* OSCER clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.25.3.6 enum clock\_flexio\_src\_kl27z4\_t

Enumerator

*kClockFlexioSrcNone* Clock disabled.  
*kClockFlexioSrcIrc48M* MCGPCLK/IRC48M.  
*kClockFlexioSrcOsc0erClk* OSCERCLK.  
*kClockFlexioSrcMcgIrClk* MCGIRCLK.

### 53.2.25.3.7 enum clock\_lptmr\_src\_kl27z4\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

## SIM HAL driver

### 53.2.25.3.8 enum clock\_clkout\_src\_kl27z4\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelOsc0erClk* OSCER clock.  
*kClockClkoutSelIrc48M* IRC48M clock.

### 53.2.25.3.9 enum clock\_rtcout\_src\_kl27z4\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrcOsc0erClk* OSCER clock.

### 53.2.25.3.10 enum sim\_adc\_pretrg\_sel\_kl27z4\_t

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.25.3.11 enum sim\_adc\_trg\_sel\_kl27z4\_t

Enumerator

*kSimAdcTrgselExt* External trigger.  
*kSimAdcTrgSelComp0* CMP0 output.  
*kSimAdcTrgSelPit0* PIT trigger 0.  
*kSimAdcTrgSelPit1* PIT trigger 1.  
*kSimAdcTrgSelTpm0* TPM0 overflow.  
*kSimAdcTrgSelTpm1* TPM1 overflow.  
*kSimAdcTrgSelTpm2* TPM2 overflow.  
*kSimAdcTrgSelRtcAlarm* RTC alarm.  
*kSimAdcTrgSelRtcSec* RTC seconds.  
*kSimAdcTrgSelLptimer* Low-power timer trigger.

### 53.2.25.3.12 enum sim\_lpuart\_rxsrc\_kl27z4\_t

Enumerator

*kSimLpuartRxsrcPin* LPUARTx\_RX Pin.  
*kSimLpuartRxsrcCmp0* CMP0.

### 53.2.25.3.13 enum sim\_lpuart\_txsrc\_kl27z4\_t

Enumerator

- kSimLpuartTxsrcPin* UARTx\_TX Pin.
- kSimLpuartTxsrcTpm1* UARTx\_TX pin modulated with TPM1 channel 0 output.
- kSimLpuartTxsrcTpm2* UARTx\_TX pin modulated with TPM2 channel 0 output.

### 53.2.25.3.14 enum clock\_sai\_src\_kl27z4\_t

Enumerator

- kClockSaiSrcSysClk* SYSCLK.
- kClockSaiSrcOsc0erClk* OSC0ERCLK.
- kClockSaiSrcMcgIrClk* MCGIRCLK.
- kClockSaiSrcIrc48M* MCGPCLK/IRC48M.

### 53.2.25.3.15 enum sim\_tpm\_clk\_sel\_kl27z4\_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 53.2.25.3.16 enum sim\_tpm\_ch\_src\_kl27z4\_t

Enumerator

- kSimTpmChSrc0* Channel y input capture source uses 0.
- kSimTpmChSrc1* Channel y input capture source uses 1.
- kSimTpmChSrc2* Channel y input capture source uses 2.
- kSimTpmChSrc3* Channel y input capture source uses 3.

### 53.2.25.3.17 enum sim\_clock\_gate\_name\_kl27z4\_t

## SIM HAL driver

### 53.2.26 KL27Z644 SIM HAL driver

#### 53.2.26.1 Overview

The section describes the enumerations, macros and data structures for KL27Z644 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKL27Z644.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_cop\\_src\\_kl27z644\\_t](#) {  
  [kClockCopSrcLpoClk](#),  
  [kClockCopSrcMcgIrClk](#),  
  [kClockCopSrcOsc0erClk](#),  
  [kClockCopSrcBusClk](#) }  
*COP clock source selection.*
- enum [clock\\_er32k\\_src\\_kl27z644\\_t](#) {  
  [kClockEr32kSrcOsc0](#) = 0U,  
  [kClockEr32kSrcRtc](#) = 2U,  
  [kClockEr32kSrcLpo](#) = 3U }  
*SIM external reference clock source select (OSC32KSEL).*
- enum [clock\\_osc32kout\\_sel\\_kl27z644\\_t](#) {  
  [kClockOsc32koutNone](#),  
  [kClockOsc32koutPte0](#) }  
*SIM external reference clock output pin select (OSC32KOUT).*
- enum [clock\\_lpuart\\_src\\_kl27z644\\_t](#) {  
  [kClockLpuartSrcNone](#),  
  [kClockLpuartSrcIrc48M](#),  
  [kClockLpuartSrcOsc0erClk](#),  
  [kClockLpuartSrcMcgIrClk](#) }  
*SIM LPUART clock source.*
- enum [clock\\_tpm\\_src\\_kl27z644\\_t](#) {  
  [kClockTpmSrcNone](#),  
  [kClockTpmSrcIrc48M](#),  
  [kClockTpmSrcOsc0erClk](#),  
  [kClockTpmSrcMcgIrClk](#) }  
*SIM TPM clock source.*

- enum `clock_usbfs_src_kl27z644_t` {  
`kClockUsbfsSrcExt`,  
`kClockUsbfsSrcIrc48M` }  
*SIM USB FS clock source.*
- enum `clock_flexio_src_kl27z644_t` {  
`kClockFlexioSrcNone`,  
`kClockFlexioSrcIrc48M`,  
`kClockFlexioSrcOsc0erClk`,  
`kClockFlexioSrcMcgIrClk` }  
*FLEXIO clock source.*
- enum `clock_lptmr_src_kl27z644_t` {  
`kClockLptmrSrcMcgIrClk`,  
`kClockLptmrSrcLpoClk`,  
`kClockLptmrSrcEr32kClk`,  
`kClockLptmrSrcOsc0erClk` }  
*LPTMR clock source select.*
- enum `clock_clkout_src_kl27z644_t` {  
`kClockClkoutSelFlashClk` = 2U,  
`kClockClkoutSelLpoClk` = 3U,  
`kClockClkoutSelMcgIrClk` = 4U,  
`kClockClkoutSelOsc0erClk` = 6U,  
`kClockClkoutSelIrc48M` = 7U }  
*SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_kl27z644_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrcOsc0erClk` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_kl27z644_t` {  
`kSimAdcPretrgselA`,  
`kSimAdcPretrgselB` }  
*SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kl27z644_t` {  
`kSimAdcTrgSelExt` = 0U,  
`kSimAdcTrgSelComp0` = 1U,  
`kSimAdcTrgSelPit0` = 4U,  
`kSimAdcTrgSelPit1` = 5U,  
`kSimAdcTrgSelTpm0` = 8U,  
`kSimAdcTrgSelTpm1` = 9U,  
`kSimAdcTrgSelTpm2` = 10U,  
`kSimAdcTrgSelRtcAlarm` = 12U,  
`kSimAdcTrgSelRtcSec` = 13U,  
`kSimAdcTrgSelLptimer` = 14U }  
*SIM ADCx trigger select.*
- enum `sim_lpuart_rxs_src_kl27z644_t` {  
`kSimLpuartRxsSrcPin`,  
`kSimLpuartRxsSrcCmp0` }

## SIM HAL driver

- LPUART receive data source.*

  - enum `sim_lpuart_txsrc_kl27z644_t` {  
`kSimLpuartTxsrcPin`,  
`kSimLpuartTxsrcTpm1`,  
`kSimLpuartTxsrcTpm2` }
- LPUART transmit data source.*

  - enum `sim_tpm_clk_sel_kl27z644_t` {  
`kSimTpmClkSel0`,  
`kSimTpmClkSel1` }
- SIM Timer/PWM external clock select.*

  - enum `sim_tpm_ch_src_kl27z644_t` {  
`kSimTpmChSrc0`,  
`kSimTpmChSrc1`,  
`kSimTpmChSrc2`,  
`kSimTpmChSrc3` }
- SIM Timer/PWM x channel y input capture source select.*

  - enum `sim_clock_gate_name_kl27z644_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.26.2 Macro Definition Documentation

53.2.26.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.26.3 Enumeration Type Documentation

#### 53.2.26.3.1 enum `clock_cop_src_kl27z644_t`

Enumerator

*kClockCopSrcLpoClk* LPO clock 1K HZ.  
*kClockCopSrcMcgIrClk* MCG IRC Clock.  
*kClockCopSrcOsc0erClk* OSCER Clock.  
*kClockCopSrcBusClk* BUS clock.

#### 53.2.26.3.2 enum `clock_er32k_src_kl27z644_t`

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.26.3.3 enum clock\_osc32kout\_sel\_kl27z644\_t

Enumerator

*kClockOsc32koutNone* ERCLK32K is not output.  
*kClockOsc32koutPte0* ERCLK32K output on PTE0.

### 53.2.26.3.4 enum clock\_lpuart\_src\_kl27z644\_t

Enumerator

*kClockLpuartSrcNone* disabled  
*kClockLpuartSrcIrc48M* IRC48M.  
*kClockLpuartSrcOsc0erClk* OSCER clock.  
*kClockLpuartSrcMcgIrClk* MCGIR clock.

### 53.2.26.3.5 enum clock\_tpm\_src\_kl27z644\_t

Enumerator

*kClockTpmSrcNone* disabled  
*kClockTpmSrcIrc48M* IRC48M/MCGPCLK.  
*kClockTpmSrcOsc0erClk* OSCER clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.26.3.6 enum clock\_usbfs\_src\_kl27z644\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)  
*kClockUsbfsSrcIrc48M* IRC48/MCGPCLK.

### 53.2.26.3.7 enum clock\_flexio\_src\_kl27z644\_t

Enumerator

*kClockFlexioSrcNone* Clock disabled.  
*kClockFlexioSrcIrc48M* MCGPCLK/IRC48M.  
*kClockFlexioSrcOsc0erClk* OSCERCLK.  
*kClockFlexioSrcMcgIrClk* MCGIRCLK.

## SIM HAL driver

### 53.2.26.3.8 enum clock\_lptmr\_src\_kl27z644\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.26.3.9 enum clock\_clkout\_src\_kl27z644\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelOsc0erClk* OSCER clock.  
*kClockClkoutSelIrc48M* IRC48M clock.

### 53.2.26.3.10 enum clock\_rtcout\_src\_kl27z644\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrcOsc0erClk* OSCER clock.

### 53.2.26.3.11 enum sim\_adc\_pretrg\_sel\_kl27z644\_t

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.26.3.12 enum sim\_adc\_trg\_sel\_kl27z644\_t

Enumerator

*kSimAdcTrgSelExt* External trigger.  
*kSimAdcTrgSelComp0* CMP0 output.  
*kSimAdcTrgSelPit0* PIT trigger 0.  
*kSimAdcTrgSelPit1* PIT trigger 1.  
*kSimAdcTrgSelTpm0* TPM0 overflow.  
*kSimAdcTrgSelTpm1* TPM1 overflow.



*kSimAdcTrgSelTpm2* TPM2 overflow.  
*kSimAdcTrgSelRtcAlarm* RTC alarm.  
*kSimAdcTrgSelRtcSec* RTC seconds.  
*kSimAdcTrgSelLptimer* Low-power timer trigger.

#### 53.2.26.3.13 enum sim\_lpuart\_rxsrc\_kl27z644\_t

Enumerator

*kSimLpuartRxsrcPin* LPUARTx\_RX Pin.  
*kSimLpuartRxsrcCmp0* CMP0.

#### 53.2.26.3.14 enum sim\_lpuart\_txsrc\_kl27z644\_t

Enumerator

*kSimLpuartTxsrcPin* UARTx\_TX Pin.  
*kSimLpuartTxsrcTpm1* UARTx\_TX pin modulated with TPM1 channel 0 output.  
*kSimLpuartTxsrcTpm2* UARTx\_TX pin modulated with TPM2 channel 0 output.

#### 53.2.26.3.15 enum sim\_tpm\_clk\_sel\_kl27z644\_t

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.  
*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

#### 53.2.26.3.16 enum sim\_tpm\_ch\_src\_kl27z644\_t

Enumerator

*kSimTpmChSrc0* Channel y input capture source uses 0.  
*kSimTpmChSrc1* Channel y input capture source uses 1.  
*kSimTpmChSrc2* Channel y input capture source uses 2.  
*kSimTpmChSrc3* Channel y input capture source uses 3.

#### 53.2.26.3.17 enum sim\_clock\_gate\_name\_kl27z644\_t

## SIM HAL driver

### 53.2.27 KL28T7 SIM HAL driver

#### 53.2.27.1 Overview

The section describes the enumerations, macros and data structures for KL28T7 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKL28T7.h](#)

#### Enumerations

- enum `clock_wdog_src_kl28t7_t` {  
`kClockWdogSrcLpoClk`,  
`kClockWdogSrcAltClk` }  
*WDOG clock source select.*
- enum `clock_trace_src_kl28t7_t` {  
`kClockTraceSrcMcgoutClk`,  
`kClockTraceSrcCoreClk` }  
*Debug trace clock source select.*
- enum `clock_port_filter_src_kl28t7_t` {  
`kClockPortFilterSrcBusClk`,  
`kClockPortFilterSrcLpoClk` }  
*PORTx digital input filter clock source select.*
- enum `clock_lptmr_src_kl28t7_t` {  
`kClockLptmrSrcMcgIrClk`,  
`kClockLptmrSrcLpoClk`,  
`kClockLptmrSrcEr32kClk`,  
`kClockLptmrSrcOsc0erClk` }  
*LPTMR clock source select.*
- enum `clock_sai_src_kl28t7_t` {  
`kClockSaiSrcSysClk` = 0U,  
`kClockSaiSrcOsc0erClk` = 1U,  
`kClockSaiSrcPllFllSel` = 3U }  
*SAI clock source.*
- enum `clock_pllfl_sel_kl28t7_t` {  
`kClockPllFllSelFll` = 0U,  
`kClockPllFllSelPll` = 1U,  
`kClockPllFllSelIrc48M` = 3U }  
*SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_kl28t7_t` {  
`kClockEr32kSrcOsc0` = 0U,  
`kClockEr32kSrcRtc` = 2U,  
`kClockEr32kSrcLpo` = 3U }  
*SIM external reference clock source select (OSC32KSEL).*

- enum `clock_rtcout_src_kl28t7_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrc32kHz` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `clock_osc32kout_sel_kl28t7_t` {  
`kClockOsc32koutNone` = 0U,  
`kClockOsc32koutPte0` = 1U,  
`kClockOsc32koutPte26` = 2U }  
*SIM OSC32KOUT selection.*
- enum `sim_usbsstby_mode_kl28t7_t` {  
`kSimUsbsstbyNoRegulator`,  
`kSimUsbsstbyWithRegulator` }  
*SIM USB voltage regulator in standby mode setting during stop modes.*
- enum `sim_usbvstby_mode_kl28t7_t` {  
`kSimUsbvstbyNoRegulator`,  
`kSimUsbvstbyWithRegulator` }  
*SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.*
- enum `sim_adc_pretrg_sel_kl28t7_t` {  
`kSimAdcPretrgselA`,  
`kSimAdcPretrgselB` }  
*SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kl28t7_t` {  
`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_rxsrc_kl28t7_t` {  
`kSimLpuartRxsrcPin`,  
`kSimLpuartRxsrcCmp0`,  
`kSimLpuartRxsrcCmp1` }  
*SIM LPUART RX source.*
- enum `sim_uart_rxsrc_kl28t7_t` {  
`kSimUartRxsrcPin`,  
`kSimUartRxsrcCmp0`,  
`kSimUartRxsrcCmp1` }  
*SIM UART receive data source select.*
- enum `sim_uart_txsrc_kl28t7_t` {

## SIM HAL driver

kSimUartTxsrcPin,  
kSimUartTxsrcFtm1,  
kSimUartTxsrcFtm2 }

*SIM UART transmit data source select.*

- enum `sim_ftm_trg_src_kl28t7_t` {  
kSimFtmTrgSrc0,  
kSimFtmTrgSrc1 }

*SIM FlexTimer x trigger y select.*

- enum `sim_ftm_clk_sel_kl28t7_t` {  
kSimFtmClkSel0,  
kSimFtmClkSel1 }

*SIM FlexTimer external clock select.*

- enum `sim_ftm_ch_src_kl28t7_t` {  
kSimFtmChSrc0,  
kSimFtmChSrc1,  
kSimFtmChSrc2,  
kSimFtmChSrc3 }

*SIM FlexTimer x channel y input capture source select.*

- enum `sim_ftm_ch_out_src_kl28t7_t` {  
kSimFtmChOutSrc0,  
kSimFtmChOutSrc1 }

*SIM FlexTimer x channel y output source select.*

- enum `sim_ftmflt_sel_kl28t7_t` {  
kSimFtmFltSel0,  
kSimFtmFltSel1 }

*SIM FlexTimer x Fault y select.*

- enum `sim_tpm_clk_sel_kl28t7_t` {  
kSimTpmClkSel0,  
kSimTpmClkSel1 }

*SIM Timer/PWM external clock select.*

- enum `sim_tpm_ch_src_kl28t7_t` {  
kSimTpmChSrc0,  
kSimTpmChSrc1 }

*SIM Timer/PWM x channel y input capture source select.*

- enum `sim_cmtuartpad_strenght_kl28t7_t` {  
kSimCmtuartSinglePad,  
kSimCmtuartDualPad }

*SIM CMT/UART pad drive strength.*

- enum `sim_ptd7pad_strenght_kl28t7_t` {  
kSimPtd7padSinglePad,  
kSimPtd7padDualPad }

*SIM PTD7 pad drive strength.*

### 53.2.27.2 Enumeration Type Documentation

#### 53.2.27.2.1 enum clock\_wdog\_src\_kl28t7\_t

Enumerator

*kClockWdogSrcLpoClk* LPO.

*kClockWdogSrcAltClk* Alternative clock, for this SOC it is Bus clock.

#### 53.2.27.2.2 enum clock\_trace\_src\_kl28t7\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

#### 53.2.27.2.3 enum clock\_port\_filter\_src\_kl28t7\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.

*kClockPortFilterSrcLpoClk* LPO.

#### 53.2.27.2.4 enum clock\_lptmr\_src\_kl28t7\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.

*kClockLptmrSrcLpoClk* LPO clock.

*kClockLptmrSrcEr32kClk* ERCLK32K clock.

*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

#### 53.2.27.2.5 enum clock\_sai\_src\_kl28t7\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.

*kClockSaiSrcOsc0erClk* OSC0ERCLK.

*kClockSaiSrcPllFllSel* MCGPLLCLK.

## SIM HAL driver

### 53.2.27.2.6 enum clock\_pllfl\_sel\_kl28t7\_t

Enumerator

*kClockPlIFllSelFll* Fll clock.  
*kClockPlIFllSelPlI* PlI0 clock.  
*kClockPlIFllSelIrc48M* IRC48MCLK.

### 53.2.27.2.7 enum clock\_er32k\_src\_kl28t7\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.27.2.8 enum clock\_rtcout\_src\_kl28t7\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

### 53.2.27.2.9 enum clock\_osc32kout\_sel\_kl28t7\_t

Enumerator

*kClockOsc32koutNone* ERCLK32K is not output.  
*kClockOsc32koutPte0* ERCLK32K is output on PTE0.  
*kClockOsc32koutPte26* ERCLK32K is output on PTE26.

### 53.2.27.2.10 enum sim\_usbsstby\_mode\_kl28t7\_t

Enumerator

*kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes  
*kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

### 53.2.27.2.11 enum sim\_usbvstby\_mode\_kl28t7\_t

Enumerator

*kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes  
*kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes

**53.2.27.2.12 enum sim\_adc\_pretrg\_sel\_kl28t7\_t**

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.**53.2.27.2.13 enum sim\_adc\_trg\_sel\_kl28t7\_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.**53.2.27.2.14 enum sim\_lpuart\_rxsrc\_kl28t7\_t**

Enumerator

*kSimLpuartRxsrcPin* LPUARTx\_RX Pin.*kSimLpuartRxsrcCmp0* CMP0.*kSimLpuartRxsrcCmp1* CMP1.**53.2.27.2.15 enum sim\_uart\_rxsrc\_kl28t7\_t**

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.*kSimUartRxsrcCmp0* CMP0.*kSimUartRxsrcCmp1* CMP1.

## SIM HAL driver

### 53.2.27.2.16 enum sim\_uart\_txsrc\_kl28t7\_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.

### 53.2.27.2.17 enum sim\_ftm\_trg\_src\_kl28t7\_t

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.

*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

### 53.2.27.2.18 enum sim\_ftm\_clk\_sel\_kl28t7\_t

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.

*kSimFtmClkSel1* FTM CLKIN1 pin.

### 53.2.27.2.19 enum sim\_ftm\_ch\_src\_kl28t7\_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.

### 53.2.27.2.20 enum sim\_ftm\_ch\_out\_src\_kl28t7\_t

Enumerator

*kSimFtmChOutSrc0* FlexTimer x channel y output source selection 0.

*kSimFtmChOutSrc1* FlexTimer x channel y output source selection 1.

### 53.2.27.2.21 enum sim\_ftmflt\_sel\_kl28t7\_t

Enumerator

*kSimFtmFltSel0* FlexTimer x fault y select 0.

*kSimFtmFltSel1* FlexTimer x fault y select 1.



**53.2.27.2.22 enum sim\_tpm\_clk\_sel\_kl28t7\_t**

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.**53.2.27.2.23 enum sim\_tpm\_ch\_src\_kl28t7\_t**

Enumerator

*kSimTpmChSrc0* TPMx\_CH0 signal.*kSimTpmChSrc1* CMP0 output.**53.2.27.2.24 enum sim\_cmtuartpad\_strenght\_kl28t7\_t**

Enumerator

*kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.*kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.**53.2.27.2.25 enum sim\_ptd7pad\_strenght\_kl28t7\_t**

Enumerator

*kSimPtd7padSinglePad* Single-pad drive strength for PTD7.*kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

## SIM HAL driver

### 53.2.28 KL33Z4 SIM HAL driver

#### 53.2.28.1 Overview

The section describes the enumerations, macros and data structures for KL33Z4 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKL33Z4.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_cop\\_src\\_kl33z4\\_t](#) {  
[kClockCopSrcLpoClk](#),  
[kClockCopSrcMcgIrClk](#),  
[kClockCopSrcOsc0erClk](#),  
[kClockCopSrcBusClk](#) }  
*COP clock source selection.*
- enum [clock\\_er32k\\_src\\_kl33z4\\_t](#) {  
[kClockEr32kSrcOsc0](#) = 0U,  
[kClockEr32kSrcRtc](#) = 2U,  
[kClockEr32kSrcLpo](#) = 3U }  
*SIM external reference clock source select (OSC32KSEL).*
- enum [clock\\_osc32kout\\_sel\\_kl33z4\\_t](#) {  
[kClockOsc32koutNone](#) = 0U,  
[kClockOsc32koutPte0](#) = 1U,  
[kClockOsc32koutPte26](#) = 2U }  
*SIM external reference clock output pin select (OSC32KOUT).*
- enum [clock\\_lpuart\\_src\\_kl33z4\\_t](#) {  
[kClockLpuartSrcNone](#),  
[kClockLpuartSrcIrc48M](#),  
[kClockLpuartSrcOsc0erClk](#),  
[kClockLpuartSrcMcgIrClk](#) }  
*SIM LPUART clock source.*
- enum [clock\\_tpm\\_src\\_kl33z4\\_t](#) {  
[kClockTpmSrcNone](#),  
[kClockTpmSrcIrc48M](#),  
[kClockTpmSrcOsc0erClk](#),  
[kClockTpmSrcMcgIrClk](#) }  
*SIM TPM clock source.*

- enum `clock_flexio_src_kl33z4_t` {  
`kClockFlexioSrcNone`,  
`kClockFlexioSrcIrc48M`,  
`kClockFlexioSrcOsc0erClk`,  
`kClockFlexioSrcMcgIrClk` }  
*FLEXIO clock source.*
- enum `clock_lptmr_src_kl33z4_t` {  
`kClockLptmrSrcMcgIrClk`,  
`kClockLptmrSrcLpoClk`,  
`kClockLptmrSrcEr32kClk`,  
`kClockLptmrSrcOsc0erClk` }  
*LPTMR clock source select.*
- enum `clock_clkout_src_kl33z4_t` {  
`kClockClkoutSelFlashClk` = 2U,  
`kClockClkoutSelLpoClk` = 3U,  
`kClockClkoutSelMcgIrClk` = 4U,  
`kClockClkoutSelOsc0erClk` = 6U,  
`kClockClkoutSelIrc48M` = 7U }  
*SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_kl33z4_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrcOsc0erClk` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_kl33z4_t` {  
`kSimAdcPretrgselA`,  
`kSimAdcPretrgselB` }  
*SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kl33z4_t` {  
`kSimAdcTrgselExt` = 0U,  
`kSimAdcTrgSelComp0` = 1U,  
`kSimAdcTrgSelPit0` = 4U,  
`kSimAdcTrgSelPit1` = 5U,  
`kSimAdcTrgSelTpm0` = 8U,  
`kSimAdcTrgSelTpm1` = 9U,  
`kSimAdcTrgSelTpm2` = 10U,  
`kSimAdcTrgSelRtcAlarm` = 12U,  
`kSimAdcTrgSelRtcSec` = 13U,  
`kSimAdcTrgSelLptimer` = 14U }  
*SIM ADCx trigger select.*
- enum `sim_lpuart_rxsrc_kl33z4_t` {  
`kSimLpuartRxsrcPin`,  
`kSimLpuartRxsrcCmp0` }  
*LPUART receive data source.*
- enum `sim_lpuart_txsrc_kl33z4_t` {  
`kSimLpuartTxsrcPin`,  
`kSimLpuartTxsrcTpm1`,

## SIM HAL driver

- `kSimLpuartTxsrcTpm2` }  
*LPUART transmit data source.*
- enum `clock_sai_src_kl33z4_t` {  
`kClockSaiSrcSysClk` = 0U,  
`kClockSaiSrcOsc0erClk` = 1U,  
`kClockSaiSrcMcgIrClk` = 2U,  
`kClockSaiSrcIrc48M` = 3U }
- SAI clock source.*
- enum `sim_tpm_clk_sel_kl33z4_t` {  
`kSimTpmClkSel0`,  
`kSimTpmClkSel1` }
- SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_kl33z4_t` {  
`kSimTpmChSrc0`,  
`kSimTpmChSrc1`,  
`kSimTpmChSrc2`,  
`kSimTpmChSrc3` }
- SIM Timer/PWM x channel y input capture source select.*
- enum `sim_clock_gate_name_kl33z4_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.28.2 Macro Definition Documentation

53.2.28.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.28.3 Enumeration Type Documentation

#### 53.2.28.3.1 enum `clock_cop_src_kl33z4_t`

Enumerator

`kClockCopSrcLpoClk` LPO clock 1K HZ.  
`kClockCopSrcMcgIrClk` MCG IRC Clock.  
`kClockCopSrcOsc0erClk` OSCER Clock.  
`kClockCopSrcBusClk` BUS clock.

#### 53.2.28.3.2 enum `clock_er32k_src_kl33z4_t`

Enumerator

`kClockEr32kSrcOsc0` OSC0 clock (OSC032KCLK).  
`kClockEr32kSrcRtc` RTC 32k clock .  
`kClockEr32kSrcLpo` LPO clock.

**53.2.28.3.3 enum clock\_osc32kout\_sel\_kl33z4\_t**

Enumerator

*kClockOsc32koutNone* ERCLK32K is not output.  
*kClockOsc32koutPte0* ERCLK32K is output on PTE0.  
*kClockOsc32koutPte26* ERCLK32K is output on PTE26.

**53.2.28.3.4 enum clock\_lpuart\_src\_kl33z4\_t**

Enumerator

*kClockLpuartSrcNone* disabled  
*kClockLpuartSrcIrc48M* IRC48M.  
*kClockLpuartSrcOsc0erClk* OSCER clock.  
*kClockLpuartSrcMcgIrClk* MCGIR clock.

**53.2.28.3.5 enum clock\_tpm\_src\_kl33z4\_t**

Enumerator

*kClockTpmSrcNone* disabled  
*kClockTpmSrcIrc48M* IRC48M/MCGPCLK.  
*kClockTpmSrcOsc0erClk* OSCER clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

**53.2.28.3.6 enum clock\_flexio\_src\_kl33z4\_t**

Enumerator

*kClockFlexioSrcNone* Clock disabled.  
*kClockFlexioSrcIrc48M* MCGPCLK/IRC48M.  
*kClockFlexioSrcOsc0erClk* OSCERCLK.  
*kClockFlexioSrcMcgIrClk* MCGIRCLK.

**53.2.28.3.7 enum clock\_lptmr\_src\_kl33z4\_t**

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

## SIM HAL driver

### 53.2.28.3.8 enum clock\_clkout\_src\_kl33z4\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelOsc0erClk* OSCER clock.  
*kClockClkoutSelIrc48M* IRC48M clock.

### 53.2.28.3.9 enum clock\_rtcout\_src\_kl33z4\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrcOsc0erClk* OSCER clock.

### 53.2.28.3.10 enum sim\_adc\_pretrg\_sel\_kl33z4\_t

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.28.3.11 enum sim\_adc\_trg\_sel\_kl33z4\_t

Enumerator

*kSimAdcTrgselExt* External trigger.  
*kSimAdcTrgSelComp0* CMP0 output.  
*kSimAdcTrgSelPit0* PIT trigger 0.  
*kSimAdcTrgSelPit1* PIT trigger 1.  
*kSimAdcTrgSelTpm0* TPM0 overflow.  
*kSimAdcTrgSelTpm1* TPM1 overflow.  
*kSimAdcTrgSelTpm2* TPM2 overflow.  
*kSimAdcTrgSelRtcAlarm* RTC alarm.  
*kSimAdcTrgSelRtcSec* RTC seconds.  
*kSimAdcTrgSelLptimer* Low-power timer trigger.

### 53.2.28.3.12 enum sim\_lpuart\_rxsrc\_kl33z4\_t

Enumerator

*kSimLpuartRxsrcPin* LPUARTx\_RX Pin.  
*kSimLpuartRxsrcCmp0* CMP0.

**53.2.28.3.13 enum sim\_lpuart\_txsrc\_kl33z4\_t**

Enumerator

*kSimLpuartTxsrcPin* UARTx\_TX Pin.*kSimLpuartTxsrcTpm1* UARTx\_TX pin modulated with TPM1 channel 0 output.*kSimLpuartTxsrcTpm2* UARTx\_TX pin modulated with TPM2 channel 0 output.**53.2.28.3.14 enum clock\_sai\_src\_kl33z4\_t**

Enumerator

*kClockSaiSrcSysClk* SYSCLK.*kClockSaiSrcOsc0erClk* OSC0ERCLK.*kClockSaiSrcMcgIrClk* MCGIRCLK.*kClockSaiSrcIrc48M* MCGPCLK/IRC48M.**53.2.28.3.15 enum sim\_tpm\_clk\_sel\_kl33z4\_t**

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.**53.2.28.3.16 enum sim\_tpm\_ch\_src\_kl33z4\_t**

Enumerator

*kSimTpmChSrc0* Channel y input capture source uses 0.*kSimTpmChSrc1* Channel y input capture source uses 1.*kSimTpmChSrc2* Channel y input capture source uses 2.*kSimTpmChSrc3* Channel y input capture source uses 3.**53.2.28.3.17 enum sim\_clock\_gate\_name\_kl33z4\_t**

## SIM HAL driver

### 53.2.29 KL33Z644 SIM HAL driver

The section describes the enumerations, macros and data structures for KL33Z644 SIM HAL driver.



## 53.2.30 KL34Z4 SIM HAL driver

### 53.2.30.1 Overview

The section describes the enumerations, macros and data structures for KL34Z4 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKL34Z4.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_cop\\_src\\_kl34z4\\_t](#) {  
  [kClockCopSrcLpoClk](#),  
  [kClockCopSrcAltClk](#) }  
*COP clock source select.*
- enum [clock\\_tpm\\_src\\_kl34z4\\_t](#) {  
  [kClockTpmSrcNone](#),  
  [kClockTpmSrcPllFllSel](#),  
  [kClockTpmSrcOsc0erClk](#),  
  [kClockTpmSrcMcgIrClk](#) }  
*TPM clock source select.*
- enum [clock\\_lptmr\\_src\\_kl34z4\\_t](#) {  
  [kClockLptmrSrcMcgIrClk](#),  
  [kClockLptmrSrcLpoClk](#),  
  [kClockLptmrSrcEr32kClk](#),  
  [kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_lpsci\\_src\\_kl34z4\\_t](#) {  
  [kClockLpsciSrcNone](#),  
  [kClockLpsciSrcPllFllSel](#),  
  [kClockLpsciSrcOsc0erClk](#),  
  [kClockLpsciSrcMcgIrClk](#) }  
*UART0 clock source select.*
- enum [clock\\_sai\\_src\\_kl34z4\\_t](#) {  
  [kClockSaiSrcSysClk](#) = 0U,  
  [kClockSaiSrcOsc0erClk](#) = 1U,  
  [kClockSaiSrcPllClk](#) = 3U }  
*SAI clock source.*

## SIM HAL driver

- enum `clock_pllfl1_sel_kl34z4_t` {  
`kClockPllFl1SelFl1`,  
`kClockPllFl1SelPll` }  
*SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_kl34z4_t` {  
`kClockEr32kSrcOsc0` = 0U,  
`kClockEr32kSrcReserved` = 1U,  
`kClockEr32kSrcRtc` = 2U,  
`kClockEr32kSrcLpo` = 3U }  
*SIM external reference clock source select (OSC32KSEL)*
- enum `clock_clkout_src_kl34z4_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_kl34z4_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrc32kHz` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_kl34z4_t` {  
`kSimAdcPretrgselA`,  
`kSimAdcPretrgselB` }  
*SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kl34z4_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_rxsrc_kl34z4_t` {  
`kSimUartRxsrcPin`,  
`kSimUartRxsrcCmp0` }  
*SIM UART receive data source select.*
- enum `sim_uart_txsrc_kl34z4_t` {  
`kSimUartTxsrcPin`,  
`kSimUartTxsrcTpm1`,  
`kSimUartTxsrcTpm2`,  
`kSimUartTxsrcReserved` }  
*SIM UART transmit data source select.*
- enum `sim_lpsci_rxsrc_kl34z4_t` {  
`kSimLpsciRxsrcPin`,  
`kSimLpsciRxsrcCmp0` }  
*SIM LPSCI receive data source select.*
- enum `sim_lpsci_txsrc_kl34z4_t` {  
`kSimLpsciTxsrcPin`,  
`kSimLpsciTxsrcTpm1`,  
`kSimLpsciTxsrcTpm2`,  
`kSimLpsciTxsrcReserved` }  
*SIM LPSCI transmit data source select.*
- enum `sim_tpm_clk_sel_kl34z4_t` {  
`kSimTpmClkSel0`,  
`kSimTpmClkSel1` }  
*SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_kl34z4_t` {  
`kSimTpmChSrc0`,  
`kSimTpmChSrc1`,  
`kSimTpmChSrc2`,  
`kSimTpmChSrc3` }  
*SIM Timer/PWM x channel y input capture source select.*
- enum `sim_clock_gate_name_kl34z4_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.30.2 Macro Definition Documentation

53.2.30.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.30.3 Enumeration Type Documentation

53.2.30.3.1 enum `clock_cop_src_kl34z4_t`

Enumerator

*kClockCopSrcLpoClk* LPO.

*kClockCopSrcAltClk* Alternative clock, for KL34Z4 it is Bus clock.

## SIM HAL driver

### 53.2.30.3.2 enum clock\_tpm\_src\_kl34z4\_t

Enumerator

*kClockTpmSrcNone* clock disabled  
*kClockTpmSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockTpmSrcOsc0erClk* OSCERCLK clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.30.3.3 enum clock\_lptmr\_src\_kl34z4\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.30.3.4 enum clock\_lpsci\_src\_kl34z4\_t

Enumerator

*kClockLpsciSrcNone* clock disabled  
*kClockLpsciSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockLpsciSrcOsc0erClk* OSCERCLK clock.  
*kClockLpsciSrcMcgIrClk* MCGIR clock.

### 53.2.30.3.5 enum clock\_sai\_src\_kl34z4\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.30.3.6 enum clock\_pllfl\_sel\_kl34z4\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.

### 53.2.30.3.7 enum clock\_er32k\_src\_kl34z4\_t

Enumerator

*kClockEr32kSrcOsc0* OSC 32k clock.  
*kClockEr32kSrcReserved* Reserved.  
*kClockEr32kSrcRtc* RTC 32k clock.  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.30.3.8 enum clock\_clkout\_src\_kl34z4\_t

Enumerator

*kClockClkoutReserved* Reserved.  
*kClockClkoutReserved1* Reserved.  
*kClockClkoutBusClk* Bus clock.  
*kClockClkoutLpoClk* LPO clock.  
*kClockClkoutMcgIrClk* MCG ir clock.  
*kClockClkoutReserved2* Reserved.  
*kClockClkoutOsc0erClk* OSC0ER clock.  
*kClockClkoutReserved3* Reserved.

### 53.2.30.3.9 enum clock\_rtcout\_src\_kl34z4\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32KHz clock

### 53.2.30.3.10 enum sim\_adc\_pretrg\_sel\_kl34z4\_t

Enumerator

*kSimAdcPretrgselA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

### 53.2.30.3.11 enum sim\_adc\_trg\_sel\_kl34z4\_t

Enumerator

*kSimAdcTrgselExt* External trigger.  
*kSimAdcTrgSelComp0* CMP0 output.  
*kSimAdcTrgSelReserved* Reserved.

## SIM HAL driver

*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.

### 53.2.30.3.12 enum sim\_uart\_rxsrc\_kl34z4\_t

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.  
*kSimUartRxsrcCmp0* CMP0.

### 53.2.30.3.13 enum sim\_uart\_txsrc\_kl34z4\_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.

### 53.2.30.3.14 enum sim\_lpsci\_rxsrc\_kl34z4\_t

Enumerator

*kSimLpsciRxsrcPin* LPSCIx\_RX Pin.  
*kSimLpsciRxsrcCmp0* CMP0.

### 53.2.30.3.15 enum sim\_lpsci\_txsrc\_kl34z4\_t

Enumerator

*kSimLpsciTxsrcPin* LPSCIx\_TX Pin.

*kSimLpsciTxsrcTpm1* LPSCIx\_TX pin modulated with TPM1 channel 0 output.  
*kSimLpsciTxsrcTpm2* LPSCIx\_TX pin modulated with TPM2 channel 0 output.  
*kSimLpsciTxsrcReserved* Reserved.

#### 53.2.30.3.16 enum sim\_tpm\_clk\_sel\_kl34z4\_t

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.  
*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

#### 53.2.30.3.17 enum sim\_tpm\_ch\_src\_kl34z4\_t

Enumerator

*kSimTpmChSrc0* TPMx\_CH0 signal.  
*kSimTpmChSrc1* CMP0 output.  
*kSimTpmChSrc2* Reserved.  
*kSimTpmChSrc3* USB start of frame pulse.

#### 53.2.30.3.18 enum sim\_clock\_gate\_name\_kl34z4\_t

## SIM HAL driver

### 53.2.31 KL36Z4 SIM HAL driver

#### 53.2.31.1 Overview

The section describes the enumerations, macros and data structures for KL36Z4 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKL36Z4.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_cop\\_src\\_kl36z4\\_t](#) {  
  [kClockCopSrcLpoClk](#),  
  [kClockCopSrcAltClk](#) }  
*COP clock source select.*
- enum [clock\\_tpm\\_src\\_kl36z4\\_t](#) {  
  [kClockTpmSrcNone](#),  
  [kClockTpmSrcPllFllSel](#),  
  [kClockTpmSrcOsc0erClk](#),  
  [kClockTpmSrcMcgIrClk](#) }  
*TPM clock source select.*
- enum [clock\\_lptmr\\_src\\_kl36z4\\_t](#) {  
  [kClockLptmrSrcMcgIrClk](#),  
  [kClockLptmrSrcLpoClk](#),  
  [kClockLptmrSrcEr32kClk](#),  
  [kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_lpsci\\_src\\_kl36z4\\_t](#) {  
  [kClockLpsciSrcNone](#),  
  [kClockLpsciSrcPllFllSel](#),  
  [kClockLpsciSrcOsc0erClk](#),  
  [kClockLpsciSrcMcgIrClk](#) }  
*UART0 clock source select.*
- enum [clock\\_sai\\_src\\_kl36z4\\_t](#) {  
  [kClockSaiSrcSysClk](#) = 0U,  
  [kClockSaiSrcOsc0erClk](#) = 1U,  
  [kClockSaiSrcPllClk](#) = 3U }  
*SAI clock source.*



- enum `clock_pllfl1_sel_kl36z4_t` {  
`kClockPllFl1SelFl1`,  
`kClockPllFl1SelPll` }  
*SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_kl36z4_t` {  
`kClockEr32kSrcOsc0` = 0U,  
`kClockEr32kSrcReserved` = 1U,  
`kClockEr32kSrcRtc` = 2U,  
`kClockEr32kSrcLpo` = 3U }  
*SIM external reference clock source select (OSC32KSEL)*
- enum `clock_clkout_src_kl36z4_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_kl36z4_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrc32kHz` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_kl36z4_t` {  
`kSimAdcPretrgselA`,  
`kSimAdcPretrgselB` }  
*SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kl36z4_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.*

## SIM HAL driver

- enum `sim_uart_rxsrc_kl36z4_t` {  
`kSimUartRxsrcPin`,  
`kSimUartRxsrcCmp0` }  
*SIM UART receive data source select.*
- enum `sim_uart_txsrc_kl36z4_t` {  
`kSimUartTxsrcPin`,  
`kSimUartTxsrcTpm1`,  
`kSimUartTxsrcTpm2`,  
`kSimUartTxsrcReserved` }  
*SIM UART transmit data source select.*
- enum `sim_lpsci_rxsrc_kl36z4_t` {  
`kSimLpsciRxsrcPin`,  
`kSimLpsciRxsrcCmp0` }  
*SIM LPSCI receive data source select.*
- enum `sim_lpsci_txsrc_kl36z4_t` {  
`kSimLpsciTxsrcPin`,  
`kSimLpsciTxsrcTpm1`,  
`kSimLpsciTxsrcTpm2`,  
`kSimLpsciTxsrcReserved` }  
*SIM LPSCI transmit data source select.*
- enum `sim_tpm_clk_sel_kl36z4_t` {  
`kSimTpmClkSel0`,  
`kSimTpmClkSel1` }  
*SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_kl36z4_t` {  
`kSimTpmChSrc0`,  
`kSimTpmChSrc1`,  
`kSimTpmChSrc2`,  
`kSimTpmChSrc3` }  
*SIM Timer/PWM x channel y input capture source select.*
- enum `sim_clock_gate_name_kl36z4_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.31.2 Macro Definition Documentation

53.2.31.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.31.3 Enumeration Type Documentation

53.2.31.3.1 `enum clock_cop_src_kl36z4_t`

Enumerator

*kClockCopSrcLpoClk* LPO.

*kClockCopSrcAltClk* Alternative clock, for KL36Z4 it is Bus clock.

### 53.2.31.3.2 enum clock\_tpm\_src\_kl36z4\_t

Enumerator

*kClockTpmSrcNone* clock disabled  
*kClockTpmSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockTpmSrcOsc0erClk* OSCERCLK clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.31.3.3 enum clock\_lptmr\_src\_kl36z4\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.31.3.4 enum clock\_lpsci\_src\_kl36z4\_t

Enumerator

*kClockLpsciSrcNone* clock disabled  
*kClockLpsciSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockLpsciSrcOsc0erClk* OSCERCLK clock.  
*kClockLpsciSrcMcgIrClk* MCGIR clock.

### 53.2.31.3.5 enum clock\_sai\_src\_kl36z4\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.31.3.6 enum clock\_pllfl\_sel\_kl36z4\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.

## SIM HAL driver

### 53.2.31.3.7 enum clock\_er32k\_src\_kl36z4\_t

Enumerator

*kClockEr32kSrcOsc0* OSC 32k clock.  
*kClockEr32kSrcReserved* Reserved.  
*kClockEr32kSrcRtc* RTC 32k clock.  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.31.3.8 enum clock\_clkout\_src\_kl36z4\_t

Enumerator

*kClockClkoutReserved* Reserved.  
*kClockClkoutReserved1* Reserved.  
*kClockClkoutBusClk* Bus clock.  
*kClockClkoutLpoClk* LPO clock.  
*kClockClkoutMcgIrClk* MCG ir clock.  
*kClockClkoutReserved2* Reserved.  
*kClockClkoutOsc0erClk* OSC0ER clock.  
*kClockClkoutReserved3* Reserved.

### 53.2.31.3.9 enum clock\_rtcout\_src\_kl36z4\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32KHz clock

### 53.2.31.3.10 enum sim\_adc\_pretrg\_sel\_kl36z4\_t

Enumerator

*kSimAdcPretrgselA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

### 53.2.31.3.11 enum sim\_adc\_trg\_sel\_kl36z4\_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.

#### 53.2.31.3.12 enum sim\_uart\_rxsrc\_kl36z4\_t

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.  
*kSimUartRxsrcCmp0* CMP0.

#### 53.2.31.3.13 enum sim\_uart\_txsrc\_kl36z4\_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.

#### 53.2.31.3.14 enum sim\_lpsci\_rxsrc\_kl36z4\_t

Enumerator

*kSimLpsciRxsrcPin* LPSCIx\_RX Pin.  
*kSimLpsciRxsrcCmp0* CMP0.

#### 53.2.31.3.15 enum sim\_lpsci\_txsrc\_kl36z4\_t

Enumerator

*kSimLpsciTxsrcPin* LPSCIx\_TX Pin.

## SIM HAL driver

*kSimLpsciTxsrcTpm1* LPSCIx\_TX pin modulated with TPM1 channel 0 output.

*kSimLpsciTxsrcTpm2* LPSCIx\_TX pin modulated with TPM2 channel 0 output.

*kSimLpsciTxsrcReserved* Reserved.

### 53.2.31.3.16 enum sim\_tpm\_clk\_sel\_kl36z4\_t

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.

*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 53.2.31.3.17 enum sim\_tpm\_ch\_src\_kl36z4\_t

Enumerator

*kSimTpmChSrc0* TPMx\_CH0 signal.

*kSimTpmChSrc1* CMP0 output.

*kSimTpmChSrc2* Reserved.

*kSimTpmChSrc3* USB start of frame pulse.

### 53.2.31.3.18 enum sim\_clock\_gate\_name\_kl36z4\_t

## 53.2.32 KL46Z4 SIM HAL driver

### 53.2.32.1 Overview

The section describes the enumerations, macros and data structures for KL46Z4 SIM HAL driver. KL46Z4 SIM code is shared by KL16Z4, KL26Z4, KL36Z4 and KL46Z4.

#### 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](#),  
  [kClockTpmSrcPIIFllSel](#),  
  [kClockTpmSrcOsc0erClk](#),  
  [kClockTpmSrcMcgIrClk](#) }  
*TPM clock source select.*
- enum [clock\\_lptmr\\_src\\_kl46z4\\_t](#) {  
  [kClockLptmrSrcMcgIrClk](#),  
  [kClockLptmrSrcLpoClk](#),  
  [kClockLptmrSrcEr32kClk](#),  
  [kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_lpsci\\_src\\_kl46z4\\_t](#) {  
  [kClockLpsciSrcNone](#),  
  [kClockLpsciSrcPIIFllSel](#),  
  [kClockLpsciSrcOsc0erClk](#),  
  [kClockLpsciSrcMcgIrClk](#) }  
*UART0 clock source select.*
- enum [clock\\_sai\\_src\\_kl46z4\\_t](#) {  
  [kClockSaiSrcSysClk](#) = 0U,  
  [kClockSaiSrcOsc0erClk](#) = 1U,  
  [kClockSaiSrcPIIClk](#) = 3U }  
*SAI clock source.*

## SIM HAL driver

- enum `clock_pllfl1_sel_kl46z4_t` {  
`kClockPllFl1SelFl1`,  
`kClockPllFl1SelPll` }  
*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_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_rxsrc_kl46z4_t` {  
`kSimUartRxsrcPin`,  
`kSimUartRxsrcCmp0` }  
*SIM UART receive data source select.*
- enum `sim_uart_txsrc_kl46z4_t` {  
`kSimUartTxsrcPin`,  
`kSimUartTxsrcTpm1`,  
`kSimUartTxsrcTpm2`,  
`kSimUartTxsrcReserved` }  
*SIM UART transmit data source select.*
- enum `sim_lpsci_rxsrc_kl46z4_t` {  
`kSimLpsciRxsrcPin`,  
`kSimLpsciRxsrcCmp0` }  
*SIM LPSCI receive data source select.*
- enum `sim_lpsci_txsrc_kl46z4_t` {  
`kSimLpsciTxsrcPin`,  
`kSimLpsciTxsrcTpm1`,  
`kSimLpsciTxsrcTpm2`,  
`kSimLpsciTxsrcReserved` }  
*SIM LPSCI 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.*

## 53.2.32.2 Macro Definition Documentation

53.2.32.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

## 53.2.32.3 Enumeration Type Documentation

53.2.32.3.1 enum `clock_cop_src_kl46z4_t`

Enumerator

*kClockCopSrcLpoClk* LPO.

*kClockCopSrcAltClk* Alternative clock, for KL46Z4 it is Bus clock.

## SIM HAL driver

### 53.2.32.3.2 enum clock\_tpm\_src\_kl46z4\_t

Enumerator

*kClockTpmSrcNone* clock disabled  
*kClockTpmSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockTpmSrcOsc0erClk* OSCERCLK clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.32.3.3 enum clock\_lptmr\_src\_kl46z4\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.32.3.4 enum clock\_lpsci\_src\_kl46z4\_t

Enumerator

*kClockLpsciSrcNone* clock disabled  
*kClockLpsciSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockLpsciSrcOsc0erClk* OSCERCLK clock.  
*kClockLpsciSrcMcgIrClk* MCGIR clock.

### 53.2.32.3.5 enum clock\_sai\_src\_kl46z4\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.32.3.6 enum clock\_pllfl\_sel\_kl46z4\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.

**53.2.32.3.7 enum clock\_er32k\_src\_kl46z4\_t**

Enumerator

*kClockEr32kSrcOsc0* OSC 32k clock.  
*kClockEr32kSrcReserved* Reserved.  
*kClockEr32kSrcRtc* RTC 32k clock.  
*kClockEr32kSrcLpo* LPO clock.

**53.2.32.3.8 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.

**53.2.32.3.9 enum clock\_rtcout\_src\_kl46z4\_t**

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32KHz clock

**53.2.32.3.10 enum sim\_adc\_pretrg\_sel\_kl46z4\_t**

Enumerator

*kSimAdcPretrgselA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

**53.2.32.3.11 enum sim\_adc\_trg\_sel\_kl46z4\_t**

Enumerator

*kSimAdcTrgselExt* External trigger.  
*kSimAdcTrgSelComp0* CMP0 output.  
*kSimAdcTrgSelReserved* Reserved.

## SIM HAL driver

*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.

### 53.2.32.3.12 enum sim\_uart\_rxsrc\_kl46z4\_t

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.  
*kSimUartRxsrcCmp0* CMP0.

### 53.2.32.3.13 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.

### 53.2.32.3.14 enum sim\_lpsci\_rxsrc\_kl46z4\_t

Enumerator

*kSimLpsciRxsrcPin* LPSCIx\_RX Pin.  
*kSimLpsciRxsrcCmp0* CMP0.

### 53.2.32.3.15 enum sim\_lpsci\_txsrc\_kl46z4\_t

Enumerator

*kSimLpsciTxsrcPin* LPSCIx\_TX Pin.

*kSimLpsciTxsrcTpm1* LPSCIx\_TX pin modulated with TPM1 channel 0 output.  
*kSimLpsciTxsrcTpm2* LPSCIx\_TX pin modulated with TPM2 channel 0 output.  
*kSimLpsciTxsrcReserved* Reserved.

#### 53.2.32.3.16 enum sim\_tpm\_clk\_sel\_kl46z4\_t

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.  
*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

#### 53.2.32.3.17 enum sim\_tpm\_ch\_src\_kl46z4\_t

Enumerator

*kSimTpmChSrc0* TPMx\_CH0 signal.  
*kSimTpmChSrc1* CMP0 output.  
*kSimTpmChSrc2* Reserved.  
*kSimTpmChSrc3* USB start of frame pulse.

#### 53.2.32.3.18 enum sim\_clock\_gate\_name\_kl46z4\_t

## SIM HAL driver

### 53.2.33 K24F25612 SIM HAL driver

#### 53.2.33.1 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](#),  
  [kClockUsbfsSrcPIIFILSel](#) }  
*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_pllflr_sel_k24f25612_t` {
  - `kClockPllFlrSelFlr = 0U,`
  - `kClockPllFlrSelPll = 1U,`
  - `kClockPllFlrSelIrc48M = 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` {
  - `kSimUsbsstbyNoRegulator,`
  - `kSimUsbsstbyWithRegulator }`
  - 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 UART receive data source select.*
- enum sim\_uart\_txsrc\_k24f25612\_t {
  - kSimUartTxsrcPin,
  - kSimUartTxsrcFtm1,
  - kSimUartTxsrcFtm2 }
- SIM UART 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\_ftmflt\_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_strength_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.*

### 53.2.33.2 Macro Definition Documentation

53.2.33.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.33.3 Enumeration Type Documentation

53.2.33.3.1 enum `clock_wdog_src_k24f25612_t`

Enumerator

*kClockWdogSrcLpoClk* LPO.

*kClockWdogSrcAltClk* Alternative clock, for this SOC it is Bus clock.

53.2.33.3.2 enum `clock_trace_src_k24f25612_t`

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

53.2.33.3.3 enum `clock_port_filter_src_k24f25612_t`

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.

*kClockPortFilterSrcLpoClk* LPO.

53.2.33.3.4 enum `clock_lptmr_src_k24f25612_t`

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.

## SIM HAL driver

*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.33.3.5 enum clock\_usbfs\_src\_k24f25612\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)  
*kClockUsbfsSrcPllFllSel* Clock divider USB FS clock.

### 53.2.33.3.6 enum clock\_flexcan\_src\_k24f25612\_t

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.

### 53.2.33.3.7 enum clock\_sai\_src\_k24f25612\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.33.3.8 enum clock\_pllfl\_sel\_k24f25612\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.  
*kClockPllFllSelIrc48M* IRC48MCLK.

### 53.2.33.3.9 enum clock\_er32k\_src\_k24f25612\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

**53.2.33.3.10 enum clock\_clkout\_src\_k24f25612\_t**

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCGIRCLK.  
*kClockClkoutSelRtc* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSC0ERCLK.  
*kClockClkoutSelIrc48M* IRC48MCLK.

**53.2.33.3.11 enum clock\_rtcout\_src\_k24f25612\_t**

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

**53.2.33.3.12 enum sim\_usbsstby\_mode\_k24f25612\_t**

Enumerator

*kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes  
*kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

**53.2.33.3.13 enum sim\_usbvstby\_mode\_k24f25612\_t**

Enumerator

*kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes  
*kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes

**53.2.33.3.14 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

### 53.2.33.3.15 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.

### 53.2.33.3.16 enum sim\_uart\_rxsrc\_k24f25612\_t

Enumerator

- kSimUartRxsrcPin* UARTx\_RX Pin.
- kSimUartRxsrcCmp0* CMP0.
- kSimUartRxsrcCmp1* CMP1.

### 53.2.33.3.17 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.

### 53.2.33.3.18 enum sim\_ftm\_trg\_src\_k24f25612\_t

Enumerator

- kSimFtmTrgSrc0* FlexTimer x trigger y select 0.
- kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

**53.2.33.3.19 enum sim\_ftm\_clk\_sel\_k24f25612\_t**

Enumerator

- kSimFtmClkSel0* FTM CLKIN0 pin.
- kSimFtmClkSel1* FTM CLKIN1 pin.

**53.2.33.3.20 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.

**53.2.33.3.21 enum sim\_ftmflt\_sel\_k24f25612\_t**

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

**53.2.33.3.22 enum sim\_tpm\_clk\_sel\_k24f25612\_t**

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

**53.2.33.3.23 enum sim\_tpm\_ch\_src\_k24f25612\_t**

Enumerator

- kSimTpmChSrc0* TPMx\_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

**53.2.33.3.24 enum sim\_cmtuartpad\_strength\_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.

---

## SIM HAL driver

53.2.33.3.25 enum sim\_clock\_gate\_name\_k24f25612\_t

## 53.2.34 K26F18 SIM HAL driver

### 53.2.34.1 Overview

The section describes the enumerations, macros and data structures for K26F18 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MK26F18.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_k26f18\\_t](#) {  
  [kClockWdogSrcLpoClk](#),  
  [kClockWdogSrcAltClk](#) }  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_k26f18\\_t](#) {  
  [kClockTraceSrcMcgoutClkDiv](#),  
  [kClockTraceSrcCoreClk](#) }  
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_k26f18\\_t](#) {  
  [kClockPortFilterSrcBusClk](#),  
  [kClockPortFilterSrcLpoClk](#) }  
*PORTx digital input filter clock source select.*
- enum [clock\\_tpm\\_src\\_k26f18\\_t](#) {  
  [kClockTpmSrcNone](#),  
  [kClockTpmSrcPlIFllSelDiv](#),  
  [kClockTpmSrcOsc0erClk](#),  
  [kClockTpmSrcMcgIrClk](#) }  
*TPM clock source select.*
- enum [clock\\_lptmr\\_src\\_k26f18\\_t](#) {  
  [kClockLptmrSrcMcgIrClk](#),  
  [kClockLptmrSrcLpoClk](#),  
  [kClockLptmrSrcEr32kClk](#),  
  [kClockLptmrSrcOsc0erClkUndiv](#) }  
*LPTMR clock source select.*
- enum [clock\\_lpuart\\_src\\_k26f18\\_t](#) {  
  [kClockLpuartSrcNone](#),  
  [kClockLpuartSrcPlIFllSel](#),  
  [kClockLpuartSrcOsc0erClk](#),

## SIM HAL driver

- `kClockLpuartSrcMcgIrClk }`
  - SIM LPUART clock source.*
- enum `sim_lpuart_rxsrc_k26f18_t` {
  - `kSimLpuartRxsSrcPin,`
  - `kSimLpuartRxsSrcCmp0,`
  - `kSimLpuartRxsSrcCmp1 }`
  - SIM LPUART RX source.*
- enum `sim_lpuart_txsrc_k26f18_t` {
  - `kSimLpuartTxsrcPin,`
  - `kSimLpuartTxsrcTpm1,`
  - `kSimLpuartTxsrcTpm2 }`
  - SIM LPUART TX source.*
- enum `clock_usbfs_src_k26f18_t` {
  - `kClockUsbfsSrcExt,`
  - `kClockUsbfsSrcPllFllSel }`
  - SIM USB FS clock source.*
- enum `clock_flexcan_src_k26f18_t` {
  - `kClockFlexcanSrcOsc0erClk,`
  - `kClockFlexcanSrcBusClk }`
  - FLEXCAN clock source select.*
- enum `clock_sdhc_src_k26f18_t` {
  - `kClockSdhcSrcCoreSysClk,`
  - `kClockSdhcSrcPllFllSel,`
  - `kClockSdhcSrcOsc0erClk,`
  - `kClockSdhcSrcExt }`
  - SDHC clock source.*
- enum `clock_sai_src_k26f18_t` {
  - `kClockSaiSrcSysClk = 0U,`
  - `kClockSaiSrcOsc0erClk = 1U,`
  - `kClockSaiSrcPllFllSel = 3U }`
  - SAI clock source.*
- enum `clock_pllfl_sel_k26f18_t` {
  - `kClockPllFllSelFll = 0U,`
  - `kClockPllFllSelPll = 1U,`
  - `kClockPllFllSelUsb1pfd = 2U,`
  - `kClockPllFllSelIrc48M = 3U }`
  - SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_k26f18_t` {
  - `kClockEr32kSrcOsc0 = 0U,`
  - `kClockEr32kSrcRtc = 2U,`
  - `kClockEr32kSrcLpo = 3U }`
  - SIM external reference clock source select (OSC32KSEL).*
- enum `clock_clkout_src_k26f18_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\_k26f18\_t {
  - kClockRtcoutSrc1Hz,
  - kClockRtcoutSrc32kHz }
  - SIM RTCCLKOUTSEL clock source select.*
  - enum clock\_usbhs\_slowclk\_src\_k26f18\_t {
    - kClockUsbhsSlowClkSrcMcgIrClk,
    - kClockUsbhsSlowClkSrcRtc32kHz }
    - SIM USBHS/USBPHY slow clock source select.*
    - enum sim\_usbstby\_mode\_k26f18\_t {
      - kSimUsbstbyNoRegulator,
      - kSimUsbstbyWithRegulator }
      - SIM USB voltage regulator in standby mode setting during stop modes.*
      - enum sim\_usbvstby\_mode\_k26f18\_t {
        - kSimUsbvstbyNoRegulator,
        - kSimUsbvstbyWithRegulator }
        - SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.*
        - enum sim\_usbvout\_mode\_k26f18\_t {
          - kSimUsbvout2\_733V,
          - kSimUsbvout3\_020V,
          - kSimUsbvout3\_074V,
          - kSimUsbvout3\_130V,
          - kSimUsbvout3\_188V,
          - kSimUsbvout3\_248V,
          - kSimUsbvout3\_310V }
          - SIM USB voltage regulator 3.3 output target.*
          - enum sim\_adc\_pretrg\_sel\_k26f18\_t {
            - kSimAdcPretrgselA,
            - kSimAdcPretrgselB }
            - SIM ADCx pre-trigger select.*
            - enum sim\_adc\_trg\_sel\_k26f18\_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,
- kSimAdcTrgSelTpm = 15U }
- SIM ADCx trigger select.*
- enum sim\_uart\_rxsrc\_k26f18\_t {
  - kSimUartRxsrcPin,
  - kSimUartRxsrcCmp0,
  - kSimUartRxsrcCmp1 }
- SIM UART receive data source select.*
- enum sim\_uart\_txsrc\_k26f18\_t {
  - kSimUartTxsrcPin,
  - kSimUartTxsrcFtm1,
  - kSimUartTxsrcFtm2 }
- SIM UART transmit data source select.*
- enum sim\_ftm\_trg\_src\_k26f18\_t {
  - kSimFtmTrgSrc0,
  - kSimFtmTrgSrc1 }
- SIM FlexTimer x trigger y select.*
- enum sim\_ftm\_clk\_sel\_k26f18\_t {
  - kSimFtmClkSel0,
  - kSimFtmClkSel1 }
- SIM FlexTimer external clock select.*
- enum sim\_ftm\_ch\_src\_k26f18\_t {
  - kSimFtmChSrc0,
  - kSimFtmChSrc1,
  - kSimFtmChSrc2,
  - kSimFtmChSrc3 }
- SIM FlexTimer x channel y input capture source select.*
- enum sim\_ftm\_ch\_out\_src\_k26f18\_t {
  - kSimFtmChOutSrc0,
  - kSimFtmChOutSrc1 }
- SIM FlexTimer x channel y output source select.*
- enum sim\_ftm\_ft\_sel\_k26f18\_t {

- kSimFtmFltSel0,
  - kSimFtmFltSel1 }
    - SIM FlexTimer x Fault y select.*
  - enum `sim_tpm_clk_sel_k26f18_t` {
    - kSimTpmClkSel0,
    - kSimTpmClkSel1 }
      - SIM Timer/PWM external clock select.*
    - enum `sim_tpm_ch_src_k26f18_t` {
      - kSimTpmChSrc0,
      - kSimTpmChSrc1,
      - kSimTpmChSrc2,
      - kSimTpmChSrc3 }
        - SIM Timer/PWM x channel y input capture source select.*
      - enum `sim_cmtuartpad_strength_k26f18_t` {
        - kSimCmtuartSinglePad,
        - kSimCmtuartDualPad }
          - SIM CMT/UART pad drive strength.*
        - enum `sim_ptd7pad_strength_k26f18_t` {
          - kSimPtd7padSinglePad,
          - kSimPtd7padDualPad }
            - SIM PTD7 pad drive strength.*
          - enum `sim_flexbus_security_level_k26f18_t` {
            - kSimFbslLevel0,
            - kSimFbslLevel1,
            - kSimFbslLevel2,
            - kSimFbslLevel3 }
              - SIM FlexBus security level.*
            - enum `sim_clock_gate_name_k26f18_t`
              - Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.34.2 Macro Definition Documentation

53.2.34.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.34.3 Enumeration Type Documentation

53.2.34.3.1 `enum clock_wdog_src_k26f18_t`

Enumerator

*kClockWdogSrcLpoClk* LPO.

*kClockWdogSrcAltClk* Alternative clock, for this SOC it is Bus clock.

## SIM HAL driver

### 53.2.34.3.2 enum clock\_trace\_src\_k26f18\_t

Enumerator

*kClockTraceSrcMcgoutClkDiv* MCG out clock divided by the fractional divider configured by SIM\_CLKDIV4[TRACEFRAC, TRACEDIV].  
*kClockTraceSrcCoreClk* core clock

### 53.2.34.3.3 enum clock\_port\_filter\_src\_k26f18\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.  
*kClockPortFilterSrcLpoClk* LPO.

### 53.2.34.3.4 enum clock\_tpm\_src\_k26f18\_t

Enumerator

*kClockTpmSrcNone* clock disabled  
*kClockTpmSrcPllFllSelDiv* clock as selected by SOPT2[PLLFLLSEL] and divided by the fractional divider configured by SIM\_CLKDIV3[PLLFLLFRAC, PLLFLLDIV].  
*kClockTpmSrcOsc0erClk* OSCERCLK clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.34.3.5 enum clock\_lptmr\_src\_k26f18\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 53.2.34.3.6 enum clock\_lpuart\_src\_k26f18\_t

Enumerator

*kClockLpuartSrcNone* Clock disabled.  
*kClockLpuartSrcPllFllSel* Clock as selected by SOPT2[PLLFLLSEL].  
*kClockLpuartSrcOsc0erClk* OSCERCLK.  
*kClockLpuartSrcMcgIrClk* MCGIRCLK.

**53.2.34.3.7 enum sim\_lpuart\_rxsrc\_k26f18\_t**

Enumerator

*kSimLpuartRxsrcPin* LPUARTx\_RX Pin.*kSimLpuartRxsrcCmp0* CMP0.*kSimLpuartRxsrcCmp1* CMP1.**53.2.34.3.8 enum sim\_lpuart\_txsrc\_k26f18\_t**

Enumerator

*kSimLpuartTxsrcPin* UARTx\_TX Pin.*kSimLpuartTxsrcTpm1* UARTx\_TX pin modulated with TPM1 channel 0 output.*kSimLpuartTxsrcTpm2* UARTx\_TX pin modulated with TPM2 channel 0 output.**53.2.34.3.9 enum clock\_usbfs\_src\_k26f18\_t**

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)*kClockUsbfsSrcPIFllSel* Clock divider USB FS clock.**53.2.34.3.10 enum clock\_flexcan\_src\_k26f18\_t**

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.*kClockFlexcanSrcBusClk* Bus clock.**53.2.34.3.11 enum clock\_sdhc\_src\_k26f18\_t**

Enumerator

*kClockSdhcSrcCoreSysClk* Core/system clock.*kClockSdhcSrcPIFllSel* clock as selected by SOPT2[PLLFLlSEL].*kClockSdhcSrcOsc0erClk* OSCERCLK clock.*kClockSdhcSrcExt* External bypass clock (SDHC0\_CLKIN)

## SIM HAL driver

### 53.2.34.3.12 enum clock\_sai\_src\_k26f18\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllFllSel* MCGPLLFLCLK.

### 53.2.34.3.13 enum clock\_pllfl\_sel\_k26f18\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.  
*kClockPllFllSelUsb1pfd* USB1 PFD clock.  
*kClockPllFllSelIrc48M* IRC48MCLK.

### 53.2.34.3.14 enum clock\_er32k\_src\_k26f18\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.34.3.15 enum clock\_clkout\_src\_k26f18\_t

Enumerator

*kClockClkoutSelFlexbusClk* Flexbus clock.  
*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCGIRCLK.  
*kClockClkoutSelRtc* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSC0ERCLK.  
*kClockClkoutSelIrc48M* IRC48MCLK.

### 53.2.34.3.16 enum clock\_rtcout\_src\_k26f18\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

**53.2.34.3.17 enum clock\_usbhs\_slowclk\_src\_k26f18\_t**

Enumerator

*kClockUsbhsSlowClkSrcMcgIrClk* MCGIRCLK clock.  
*kClockUsbhsSlowClkSrcRtc32kHz* RTC 32kHz clock.

**53.2.34.3.18 enum sim\_usbsstby\_mode\_k26f18\_t**

Enumerator

*kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes  
*kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

**53.2.34.3.19 enum sim\_usbvstby\_mode\_k26f18\_t**

Enumerator

*kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes  
*kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes

**53.2.34.3.20 enum sim\_usbvout\_mode\_k26f18\_t**

Enumerator

*kSimUsbvout2\_733V* USB 3V regulator output voltage set to 2.733V.  
*kSimUsbvout3\_020V* USB 3V regulator output voltage set to 3.020V.  
*kSimUsbvout3\_074V* USB 3V regulator output voltage set to 3.074V.  
*kSimUsbvout3\_130V* USB 3V regulator output voltage set to 3.130V.  
*kSimUsbvout3\_188V* USB 3V regulator output voltage set to 3.188V.  
*kSimUsbvout3\_248V* USB 3V regulator output voltage set to 3.248V.  
*kSimUsbvout3\_310V* USB 3V regulator output voltage set to 3.310V.

**53.2.34.3.21 enum sim\_adc\_pretrg\_sel\_k26f18\_t**

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

## SIM HAL driver

### 53.2.34.3.22 enum sim\_adc\_trg\_sel\_k26f18\_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.
- kSimAdcTrgSelTpm* TPMx channel 0 (A pretrigger) and channel 1 (B pretrigger)

### 53.2.34.3.23 enum sim\_uart\_rxsrc\_k26f18\_t

Enumerator

- kSimUartRxsrcPin* UARTx\_RX Pin.
- kSimUartRxsrcCmp0* CMP0.
- kSimUartRxsrcCmp1* CMP1.

### 53.2.34.3.24 enum sim\_uart\_txsrc\_k26f18\_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.

### 53.2.34.3.25 enum sim\_ftm\_trg\_src\_k26f18\_t

Enumerator

- kSimFtmTrgSrc0* FlexTimer x trigger y select 0.
- kSimFtmTrgSrc1* FlexTimer x trigger y select 1.



**53.2.34.3.26 enum sim\_ftm\_clk\_sel\_k26f18\_t**

Enumerator

- kSimFtmClkSel0* FTM CLKIN0 pin.
- kSimFtmClkSel1* FTM CLKIN1 pin.

**53.2.34.3.27 enum sim\_ftm\_ch\_src\_k26f18\_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.

**53.2.34.3.28 enum sim\_ftm\_ch\_out\_src\_k26f18\_t**

Enumerator

- kSimFtmChOutSrc0* FlexTimer x channel y output source selection 0.
- kSimFtmChOutSrc1* FlexTimer x channel y output source selection 1.

**53.2.34.3.29 enum sim\_ftmflt\_sel\_k26f18\_t**

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

**53.2.34.3.30 enum sim\_tpm\_clk\_sel\_k26f18\_t**

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

**53.2.34.3.31 enum sim\_tpm\_ch\_src\_k26f18\_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.

## SIM HAL driver

### 53.2.34.3.32 enum sim\_cmtuartpad\_streng\_h\_k26f18\_t

Enumerator

*kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.

*kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.

### 53.2.34.3.33 enum sim\_ptd7pad\_streng\_h\_k26f18\_t

Enumerator

*kSimPtd7padSinglePad* Single-pad drive strength for PTD7.

*kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

### 53.2.34.3.34 enum sim\_flexbus\_security\_level\_k26f18\_t

Enumerator

*kSimFbslLevel0* FlexBus security level 0.

*kSimFbslLevel1* FlexBus security level 1.

*kSimFbslLevel2* FlexBus security level 2.

*kSimFbslLevel3* FlexBus security level 3.

### 53.2.34.3.35 enum sim\_clock\_gate\_name\_k26f18\_t

## 53.2.35 K30D10 SIM HAL driver

### 53.2.35.1 Overview

The section describes the enumerations, macros and data structures for K30D10 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MK30D10.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_k30d10\\_t](#) {  
[kClockWdogSrcLpoClk](#),  
[kClockWdogSrcAltClk](#) }  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_k30d10\\_t](#) {  
[kClockTraceSrcMcgoutClk](#),  
[kClockTraceSrcCoreClk](#) }  
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_k30d10\\_t](#) {  
[kClockPortFilterSrcBusClk](#),  
[kClockPortFilterSrcLpoClk](#) }  
*PORTx digital input filter clock source select.*
- enum [clock\\_lptmr\\_src\\_k30d10\\_t](#) {  
[kClockLptmrSrcMcgIrClk](#),  
[kClockLptmrSrcLpoClk](#),  
[kClockLptmrSrcEr32kClk](#),  
[kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_time\\_src\\_k30d10\\_t](#) {  
[kClockTimeSrcCoreSysClk](#),  
[kClockTimeSrcPIIFilSel](#),  
[kClockTimeSrcOsc0erClk](#),  
[kClockTimeSrcExt](#) }  
*SIM timestamp clock source.*
- enum [clock\\_rmii\\_src\\_k30d10\\_t](#) {  
[kClockRmiiSrcExtalClk](#),  
[kClockRmiiSrcExt](#) }  
*SIM RMII clock source.*

## SIM HAL driver

- enum `clock_flexcan_src_k30d10_t` {  
`kClockFlexcanSrcOsc0erClk`,  
`kClockFlexcanSrcBusClk` }  
*FLEXCAN clock source select.*
- enum `clock_sdhc_src_k30d10_t` {  
`kClockSdhcSrcCoreSysClk`,  
`kClockSdhcSrcPllFllSel`,  
`kClockSdhcSrcOsc0erClk`,  
`kClockSdhcSrcExt` }  
*SDHC clock source.*
- enum `clock_sai_src_k30d10_t` {  
`kClockSaiSrcSysClk` = 0U,  
`kClockSaiSrcOsc0erClk` = 1U,  
`kClockSaiSrcPllClk` = 3U }  
*SAI clock source.*
- enum `clock_tsi_active_mode_src_k30d10_t` {  
`kClockTsiActiveSrcBusClk`,  
`kClockTsiActiveSrcMcgIrClk`,  
`kClockTsiActiveSrcOsc0erClk` }  
*TSI Active Mode clock source.*
- enum `clock_tsi_lp_mode_src_k30d10_t` {  
`kClockTsiLpSrcLpoClk`,  
`kClockTsiLpSrcEr32kClk` }  
*TSI Low-power Mode clock source.*
- enum `clock_pllfl_sel_k30d10_t` {  
`kClockPllFllSelFll` = 0U,  
`kClockPllFllSelPll` = 1U }  
*SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_k30d10_t` {  
`kClockEr32kSrcOsc0` = 0U,  
`kClockEr32kSrcRtc` = 2U,  
`kClockEr32kSrcLpo` = 3U }  
*SIM external reference clock source select (OSC32KSEL).*
- enum `clock_clkout_src_k30d10_t` {  
`kClockClkoutSelFlexbusClk` = 0U,  
`kClockClkoutSelFlashClk` = 2U,  
`kClockClkoutSelLpoClk` = 3U,  
`kClockClkoutSelMcgIrClk` = 4U,  
`kClockClkoutSelRtc32kClk` = 5U,  
`kClockClkoutSelOsc0erClk` = 6U }  
*SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_k30d10_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrc32kHz` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_k30d10_t` {  
`kSimAdcPretrgSelA`,

- kSimAdcPretrgSelB }
    - SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_k30d10_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_rxsrc_k30d10_t` {
  - kSimUartRxsrcPin,
  - kSimUartRxsrcCmp0,
  - kSimUartRxsrcCmp1 }
  - SIM UART receive data source select.*
- enum `sim_uart_txsrc_k30d10_t` {
  - kSimUartTxsrcPin,
  - kSimUartTxsrcFtm1,
  - kSimUartTxsrcFtm2 }
  - SIM UART transmit data source select.*
- enum `sim_ftm_trg_src_k30d10_t` {
  - kSimFtmTrgSrc0,
  - kSimFtmTrgSrc1 }
  - SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_k30d10_t` {
  - kSimFtmClkSel0,
  - kSimFtmClkSel1 }
  - SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_k30d10_t` {
  - kSimFtmChSrc0,
  - kSimFtmChSrc1,
  - kSimFtmChSrc2,
  - kSimFtmChSrc3 }
  - SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftmflt_sel_k30d10_t` {
  - kSimFtmFltSel0,
  - kSimFtmFltSel1 }
  - SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_k30d10_t` {

## SIM HAL driver

- `kSimTpmClkSel0,`
  - `kSimTpmClkSel1 }`

*SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_k30d10_t` {
  - `kSimTpmChSrc0,`
  - `kSimTpmChSrc1 }`

*SIM Timer/PWM x channel y input capture source select.*
- enum `sim_cmtuartpad_strength_k30d10_t` {
  - `kSimCmtuartSinglePad,`
  - `kSimCmtuartDualPad }`

*SIM CMT/UART pad drive strength.*
- enum `sim_ptd7pad_strength_k30d10_t` {
  - `kSimPtd7padSinglePad,`
  - `kSimPtd7padDualPad }`

*SIM PTD7 pad drive strength.*
- enum `sim_flexbus_security_level_k30d10_t` {
  - `kSimFbslLevel0,`
  - `kSimFbslLevel1,`
  - `kSimFbslLevel2,`
  - `kSimFbslLevel3 }`

*SIM FlexBus security level.*
- enum `sim_clock_gate_name_k30d10_t`

*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.35.2 Macro Definition Documentation

53.2.35.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.35.3 Enumeration Type Documentation

#### 53.2.35.3.1 enum `clock_wdog_src_k30d10_t`

Enumerator

- `kClockWdogSrcLpoClk` LPO.
- `kClockWdogSrcAltClk` Alternative clock, for K30D10 it is Bus clock.

#### 53.2.35.3.2 enum `clock_trace_src_k30d10_t`

Enumerator

- `kClockTraceSrcMcgoutClk` MCG out clock.
- `kClockTraceSrcCoreClk` core clock

### 53.2.35.3.3 enum clock\_port\_filter\_src\_k30d10\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.  
*kClockPortFilterSrcLpoClk* LPO.

### 53.2.35.3.4 enum clock\_lptmr\_src\_k30d10\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG IRC clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.35.3.5 enum clock\_time\_src\_k30d10\_t

Enumerator

*kClockTimeSrcCoreSysClk* Core/system clock.  
*kClockTimeSrcPllFllSel* clock as selected by SOPT2[PLLFLLSSEL].  
*kClockTimeSrcOsc0erClk* OSCERCLK clock.  
*kClockTimeSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

### 53.2.35.3.6 enum clock\_rmii\_src\_k30d10\_t

Enumerator

*kClockRmiiSrcExtalClk* EXTAL Clock.  
*kClockRmiiSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

### 53.2.35.3.7 enum clock\_flexcan\_src\_k30d10\_t

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.

## SIM HAL driver

### 53.2.35.3.8 enum clock\_sdhc\_src\_k30d10\_t

Enumerator

*kClockSdhcSrcCoreSysClk* Core/system clock.  
*kClockSdhcSrcPllFllSel* clock as selected by SOPT2[PLLFLLSEL].  
*kClockSdhcSrcOsc0erClk* OSCERCLK clock.  
*kClockSdhcSrcExt* External bypass clock (SDHC0\_CLKIN)

### 53.2.35.3.9 enum clock\_sai\_src\_k30d10\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.35.3.10 enum clock\_tsi\_active\_mode\_src\_k30d10\_t

Enumerator

*kClockTsiActiveSrcBusClk* Bus clock.  
*kClockTsiActiveSrcMcgIrClk* MCG IRC clock.  
*kClockTsiActiveSrcOsc0erClk* OSCERCLK clock.

### 53.2.35.3.11 enum clock\_tsi\_lp\_mode\_src\_k30d10\_t

Enumerator

*kClockTsiLpSrcLpoClk* LPO clock.  
*kClockTsiLpSrcEr32kClk* ERCLK32K clock.

### 53.2.35.3.12 enum clock\_pllfl\_sel\_k30d10\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.



### 53.2.35.3.13 enum clock\_er32k\_src\_k30d10\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.35.3.14 enum clock\_clkout\_src\_k30d10\_t

Enumerator

*kClockClkoutSelFlexbusClk* Flexbus clock.  
*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelRtc32kClk* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSCERCLK0 clock.

### 53.2.35.3.15 enum clock\_rtcout\_src\_k30d10\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

### 53.2.35.3.16 enum sim\_adc\_pretrg\_sel\_k30d10\_t

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.35.3.17 enum sim\_adc\_trg\_sel\_k30d10\_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.

## SIM HAL driver

*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.

### 53.2.35.3.18 enum sim\_uart\_rxsrc\_k30d10\_t

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.  
*kSimUartRxsrcCmp0* CMP0.  
*kSimUartRxsrcCmp1* CMP1.

### 53.2.35.3.19 enum sim\_uart\_txsrc\_k30d10\_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.

### 53.2.35.3.20 enum sim\_ftm\_trg\_src\_k30d10\_t

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.  
*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

### 53.2.35.3.21 enum sim\_ftm\_clk\_sel\_k30d10\_t

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.  
*kSimFtmClkSel1* FTM CLKIN1 pin.

**53.2.35.3.22 enum sim\_ftm\_ch\_src\_k30d10\_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.

**53.2.35.3.23 enum sim\_ftmflt\_sel\_k30d10\_t**

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

**53.2.35.3.24 enum sim\_tpm\_clk\_sel\_k30d10\_t**

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

**53.2.35.3.25 enum sim\_tpm\_ch\_src\_k30d10\_t**

Enumerator

- kSimTpmChSrc0* TPMx\_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

**53.2.35.3.26 enum sim\_cmtuartpad\_streng\_h\_k30d10\_t**

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.

**53.2.35.3.27 enum sim\_ptd7pad\_streng\_h\_k30d10\_t**

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

## SIM HAL driver

### 53.2.35.3.28 enum sim\_flexbus\_security\_level\_k30d10\_t

Enumerator

*kSimFbslLevel0* FlexBus security level 0.

*kSimFbslLevel1* FlexBus security level 1.

*kSimFbslLevel2* FlexBus security level 2.

*kSimFbslLevel3* FlexBus security level 3.

### 53.2.35.3.29 enum sim\_clock\_gate\_name\_k30d10\_t

## 53.2.36 K40D10 SIM HAL driver

### 53.2.36.1 Overview

The section describes the enumerations, macros and data structures for K40D10 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MK40D10.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_k40d10\\_t](#) {  
[kClockWdogSrcLpoClk](#),  
[kClockWdogSrcAltClk](#) }  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_k40d10\\_t](#) {  
[kClockTraceSrcMcgoutClk](#),  
[kClockTraceSrcCoreClk](#) }  
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_k40d10\\_t](#) {  
[kClockPortFilterSrcBusClk](#),  
[kClockPortFilterSrcLpoClk](#) }  
*PORTx digital input filter clock source select.*
- enum [clock\\_lptmr\\_src\\_k40d10\\_t](#) {  
[kClockLptmrSrcMcgIrClk](#),  
[kClockLptmrSrcLpoClk](#),  
[kClockLptmrSrcEr32kClk](#),  
[kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_time\\_src\\_k40d10\\_t](#) {  
[kClockTimeSrcCoreSysClk](#),  
[kClockTimeSrcPIIFilSel](#),  
[kClockTimeSrcOsc0erClk](#),  
[kClockTimeSrcExt](#) }  
*SIM timestamp clock source.*
- enum [clock\\_rmii\\_src\\_k40d10\\_t](#) {  
[kClockRmiiSrcExtalClk](#),  
[kClockRmiiSrcExt](#) }  
*SIM RMII clock source.*

## SIM HAL driver

- enum `clock_flexcan_src_k40d10_t` {  
`kClockFlexcanSrcOsc0erClk`,  
`kClockFlexcanSrcBusClk` }  
*FLEXCAN clock source select.*
- enum `clock_sdhc_src_k40d10_t` {  
`kClockSdhcSrcCoreSysClk`,  
`kClockSdhcSrcPllFllSel`,  
`kClockSdhcSrcOsc0erClk`,  
`kClockSdhcSrcExt` }  
*SDHC clock source.*
- enum `clock_sai_src_k40d10_t` {  
`kClockSaiSrcSysClk` = 0U,  
`kClockSaiSrcOsc0erClk` = 1U,  
`kClockSaiSrcPllClk` = 3U }  
*SAI clock source.*
- enum `clock_tsi_active_mode_src_k40d10_t` {  
`kClockTsiActiveSrcBusClk`,  
`kClockTsiActiveSrcMcgIrClk`,  
`kClockTsiActiveSrcOsc0erClk` }  
*TSI Active Mode clock source.*
- enum `clock_tsi_lp_mode_src_k40d10_t` {  
`kClockTsiLpSrcLpoClk`,  
`kClockTsiLpSrcEr32kClk` }  
*TSI Low-power Mode clock source.*
- enum `clock_pllfl_sel_k40d10_t` {  
`kClockPllFllSelFll` = 0U,  
`kClockPllFllSelPll` = 1U }  
*SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_k40d10_t` {  
`kClockEr32kSrcOsc0` = 0U,  
`kClockEr32kSrcRtc` = 2U,  
`kClockEr32kSrcLpo` = 3U }  
*SIM external reference clock source select (OSC32KSEL).*
- enum `clock_clkout_src_k40d10_t` {  
`kClockClkoutSelFlexbusClk` = 0U,  
`kClockClkoutSelFlashClk` = 2U,  
`kClockClkoutSelLpoClk` = 3U,  
`kClockClkoutSelMcgIrClk` = 4U,  
`kClockClkoutSelRtc32kClk` = 5U,  
`kClockClkoutSelOsc0erClk` = 6U }  
*SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_k40d10_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrc32kHz` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_k40d10_t` {  
`kSimAdcPretrgSelA`,

- ```

kSimAdcPretrgSelB }
    SIM ADCx pre-trigger select.
• enum sim_adc_trg_sel_k40d10_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_rxsrc_k40d10_t {
    kSimUartRxsrcPin,
    kSimUartRxsrcCmp0,
    kSimUartRxsrcCmp1 }
    SIM UART receive data source select.
• enum sim_uart_txsrc_k40d10_t {
    kSimUartTxsrcPin,
    kSimUartTxsrcFtm1,
    kSimUartTxsrcFtm2 }
    SIM UART transmit data source select.
• enum sim_ftm_trg_src_k40d10_t {
    kSimFtmTrgSrc0,
    kSimFtmTrgSrc1 }
    SIM FlexTimer x trigger y select.
• enum sim_ftm_clk_sel_k40d10_t {
    kSimFtmClkSel0,
    kSimFtmClkSel1 }
    SIM FlexTimer external clock select.
• enum sim_ftm_ch_src_k40d10_t {
    kSimFtmChSrc0,
    kSimFtmChSrc1,
    kSimFtmChSrc2,
    kSimFtmChSrc3 }
    SIM FlexTimer x channel y input capture source select.
• enum sim_ftmflt_sel_k40d10_t {
    kSimFtmFltSel0,
    kSimFtmFltSel1 }
    SIM FlexTimer x Fault y select.
• enum sim_tpm_clk_sel_k40d10_t {
    
```

SIM HAL driver

- `kSimTpmClkSel0,`
 - `kSimTpmClkSel1 }`

SIM Timer/PWM external clock select.
- enum `sim_tpm_ch_src_k40d10_t` {
 - `kSimTpmChSrc0,`
 - `kSimTpmChSrc1 }`

SIM Timer/PWM x channel y input capture source select.
- enum `sim_cmtuartpad_strength_k40d10_t` {
 - `kSimCmtuartSinglePad,`
 - `kSimCmtuartDualPad }`

SIM CMT/UART pad drive strength.
- enum `sim_ptd7pad_strength_k40d10_t` {
 - `kSimPtd7padSinglePad,`
 - `kSimPtd7padDualPad }`

SIM PTD7 pad drive strength.
- enum `sim_flexbus_security_level_k40d10_t` {
 - `kSimFbslLevel0,`
 - `kSimFbslLevel1,`
 - `kSimFbslLevel2,`
 - `kSimFbslLevel3 }`

SIM FlexBus security level.
- enum `sim_clock_gate_name_k40d10_t`

Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.

53.2.36.2 Macro Definition Documentation

53.2.36.2.1 `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`

53.2.36.3 Enumeration Type Documentation

53.2.36.3.1 enum `clock_wdog_src_k40d10_t`

Enumerator

kClockWdogSrcLpoClk LPO.

kClockWdogSrcAltClk Alternative clock, for K40D10 it is Bus clock.

53.2.36.3.2 enum `clock_trace_src_k40d10_t`

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.

kClockTraceSrcCoreClk core clock

53.2.36.3.3 enum clock_port_filter_src_k40d10_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.
kClockPortFilterSrcLpoClk LPO.

53.2.36.3.4 enum clock_lptmr_src_k40d10_t

Enumerator

kClockLptmrSrcMcgIrClk MCG IRC clock.
kClockLptmrSrcLpoClk LPO clock.
kClockLptmrSrcEr32kClk ERCLK32K clock.
kClockLptmrSrcOsc0erClk OSCERCLK clock.

53.2.36.3.5 enum clock_time_src_k40d10_t

Enumerator

kClockTimeSrcCoreSysClk Core/system clock.
kClockTimeSrcPllFllSel clock as selected by SOPT2[PLLFLLSSEL].
kClockTimeSrcOsc0erClk OSCERCLK clock.
kClockTimeSrcExt ENET 1588 clock in (ENET_1588_CLKIN)

53.2.36.3.6 enum clock_rmii_src_k40d10_t

Enumerator

kClockRmiiSrcExtalClk EXTAL Clock.
kClockRmiiSrcExt ENET 1588 clock in (ENET_1588_CLKIN)

53.2.36.3.7 enum clock_flexcan_src_k40d10_t

Enumerator

kClockFlexcanSrcOsc0erClk OSCERCLK.
kClockFlexcanSrcBusClk Bus clock.

SIM HAL driver

53.2.36.3.8 enum clock_sdhc_src_k40d10_t

Enumerator

- kClockSdhcSrcCoreSysClk* Core/system clock.
- kClockSdhcSrcPllFllSel* clock as selected by SOPT2[PLLFLLSEL].
- kClockSdhcSrcOsc0erClk* OSCERCLK clock.
- kClockSdhcSrcExt* External bypass clock (SDHC0_CLKIN)

53.2.36.3.9 enum clock_sai_src_k40d10_t

Enumerator

- kClockSaiSrcSysClk* SYSCLK.
- kClockSaiSrcOsc0erClk* OSC0ERCLK.
- kClockSaiSrcPllClk* MCGPLLCLK.

53.2.36.3.10 enum clock_tsi_active_mode_src_k40d10_t

Enumerator

- kClockTsiActiveSrcBusClk* Bus clock.
- kClockTsiActiveSrcMcgIrClk* MCG IRC clock.
- kClockTsiActiveSrcOsc0erClk* OSCERCLK clock.

53.2.36.3.11 enum clock_tsi_lp_mode_src_k40d10_t

Enumerator

- kClockTsiLpSrcLpoClk* LPO clock.
- kClockTsiLpSrcEr32kClk* ERCLK32K clock.

53.2.36.3.12 enum clock_pllfl_sel_k40d10_t

Enumerator

- kClockPllFllSelFll* Fll clock.
- kClockPllFllSelPllo* Pllo clock.

53.2.36.3.13 enum clock_er32k_src_k40d10_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcRtc RTC 32k clock .
kClockEr32kSrcLpo LPO clock.

53.2.36.3.14 enum clock_clkout_src_k40d10_t

Enumerator

kClockClkoutSelFlexbusClk Flexbus clock.
kClockClkoutSelFlashClk Flash clock.
kClockClkoutSelLpoClk LPO clock.
kClockClkoutSelMcgIrClk MCG out clock.
kClockClkoutSelRtc32kClk RTC 32k clock.
kClockClkoutSelOsc0erClk OSCERCLK0 clock.

53.2.36.3.15 enum clock_rtcout_src_k40d10_t

Enumerator

kClockRtcoutSrc1Hz 1Hz clock
kClockRtcoutSrc32kHz 32kHz clock

53.2.36.3.16 enum sim_adc_pretrg_sel_k40d10_t

Enumerator

kSimAdcPretrgSelA Pre-trigger A selected for ADCx.
kSimAdcPretrgSelB Pre-trigger B selected for ADCx.

53.2.36.3.17 enum sim_adc_trg_sel_k40d10_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.

SIM HAL driver

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.

53.2.36.3.18 enum sim_uart_rxsrc_k40d10_t

Enumerator

kSimUartRxsrcPin UARTx_RX Pin.
kSimUartRxsrcCmp0 CMP0.
kSimUartRxsrcCmp1 CMP1.

53.2.36.3.19 enum sim_uart_txsrc_k40d10_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.

53.2.36.3.20 enum sim_ftm_trg_src_k40d10_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.
kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

53.2.36.3.21 enum sim_ftm_clk_sel_k40d10_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.
kSimFtmClkSel1 FTM CLKIN1 pin.

53.2.36.3.22 enum sim_ftm_ch_src_k40d10_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.

53.2.36.3.23 enum sim_ftmflt_sel_k40d10_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

53.2.36.3.24 enum sim_tpm_clk_sel_k40d10_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM_CLKIN1 pin.

53.2.36.3.25 enum sim_tpm_ch_src_k40d10_t

Enumerator

- kSimTpmChSrc0* TPMx_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

53.2.36.3.26 enum sim_cmtuartpad_streng_h_k40d10_t

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0_TXD.

53.2.36.3.27 enum sim_ptd7pad_streng_h_k40d10_t

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

SIM HAL driver

53.2.36.3.28 enum sim_flexbus_security_level_k40d10_t

Enumerator

kSimFbslLevel0 FlexBus security level 0.

kSimFbslLevel1 FlexBus security level 1.

kSimFbslLevel2 FlexBus security level 2.

kSimFbslLevel3 FlexBus security level 3.

53.2.36.3.29 enum sim_clock_gate_name_k40d10_t

53.2.37 K50D10 SIM HAL driver

53.2.37.1 Overview

The section describes the enumerations, macros and data structures for K50D10 SIM HAL driver.

Files

- file [fsl_sim_hal_MK50D10.h](#)

Macros

- #define [FSL_SIM_SCGC_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)
SIM SCGC bit index.

Enumerations

- enum [clock_wdog_src_k50d10_t](#) {
[kClockWdogSrcLpoClk](#),
[kClockWdogSrcAltClk](#) }
WDOG clock source select.
- enum [clock_trace_src_k50d10_t](#) {
[kClockTraceSrcMcgoutClk](#),
[kClockTraceSrcCoreClk](#) }
Debug trace clock source select.
- enum [clock_port_filter_src_k50d10_t](#) {
[kClockPortFilterSrcBusClk](#),
[kClockPortFilterSrcLpoClk](#) }
PORTx digital input filter clock source select.
- enum [clock_lptmr_src_k50d10_t](#) {
[kClockLptmrSrcMcgIrClk](#),
[kClockLptmrSrcLpoClk](#),
[kClockLptmrSrcEr32kClk](#),
[kClockLptmrSrcOsc0erClk](#) }
LPTMR clock source select.
- enum [clock_time_src_k50d10_t](#) {
[kClockTimeSrcCoreSysClk](#),
[kClockTimeSrcPIIFilSel](#),
[kClockTimeSrcOsc0erClk](#),
[kClockTimeSrcExt](#) }
SIM timestamp clock source.
- enum [clock_rmii_src_k50d10_t](#) {
[kClockRmiiSrcExtalClk](#),
[kClockRmiiSrcExt](#) }
SIM RMII clock source.

SIM HAL driver

- enum `clock_flexcan_src_k50d10_t` {
`kClockFlexcanSrcOsc0erClk`,
`kClockFlexcanSrcBusClk` }
FLEXCAN clock source select.
- enum `clock_sdhc_src_k50d10_t` {
`kClockSdhcSrcCoreSysClk`,
`kClockSdhcSrcPllFllSel`,
`kClockSdhcSrcOsc0erClk`,
`kClockSdhcSrcExt` }
SDHC clock source.
- enum `clock_sai_src_k50d10_t` {
`kClockSaiSrcSysClk` = 0U,
`kClockSaiSrcOsc0erClk` = 1U,
`kClockSaiSrcPllClk` = 3U }
SAI clock source.
- enum `clock_tsi_active_mode_src_k50d10_t` {
`kClockTsiActiveSrcBusClk`,
`kClockTsiActiveSrcMcgIrClk`,
`kClockTsiActiveSrcOsc0erClk` }
TSI Active Mode clock source.
- enum `clock_tsi_lp_mode_src_k50d10_t` {
`kClockTsiLpSrcLpoClk`,
`kClockTsiLpSrcEr32kClk` }
TSI Low-power Mode clock source.
- enum `clock_pllfl_sel_k50d10_t` {
`kClockPllFllSelFll` = 0U,
`kClockPllFllSelPll` = 1U }
SIM PLLFLLSEL clock source select.
- enum `clock_er32k_src_k50d10_t` {
`kClockEr32kSrcOsc0` = 0U,
`kClockEr32kSrcRtc` = 2U,
`kClockEr32kSrcLpo` = 3U }
SIM external reference clock source select (OSC32KSEL).
- enum `clock_clkout_src_k50d10_t` {
`kClockClkoutSelFlexbusClk` = 0U,
`kClockClkoutSelFlashClk` = 2U,
`kClockClkoutSelLpoClk` = 3U,
`kClockClkoutSelMcgIrClk` = 4U,
`kClockClkoutSelRtc32kClk` = 5U,
`kClockClkoutSelOsc0erClk` = 6U }
SIM CLKOUT_SEL clock source select.
- enum `clock_rtcout_src_k50d10_t` {
`kClockRtcoutSrc1Hz`,
`kClockRtcoutSrc32kHz` }
SIM RTCCLKOUTSEL clock source select.
- enum `sim_adc_pretrg_sel_k50d10_t` {
`kSimAdcPretrgSelA`,

- kSimAdcPretrgSelB }
 - SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_k50d10_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_rxsrc_k50d10_t` {
 - kSimUartRxsrcPin,
 - kSimUartRxsrcCmp0,
 - kSimUartRxsrcCmp1 }
 - SIM UART receive data source select.*
- enum `sim_uart_txsrc_k50d10_t` {
 - kSimUartTxsrcPin,
 - kSimUartTxsrcFtm1,
 - kSimUartTxsrcFtm2 }
 - SIM UART transmit data source select.*
- enum `sim_ftm_trg_src_k50d10_t` {
 - kSimFtmTrgSrc0,
 - kSimFtmTrgSrc1 }
 - SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_k50d10_t` {
 - kSimFtmClkSel0,
 - kSimFtmClkSel1 }
 - SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_k50d10_t` {
 - kSimFtmChSrc0,
 - kSimFtmChSrc1,
 - kSimFtmChSrc2,
 - kSimFtmChSrc3 }
 - SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftmflt_sel_k50d10_t` {
 - kSimFtmFltSel0,
 - kSimFtmFltSel1 }
 - SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_k50d10_t` {

SIM HAL driver

- `kSimTpmClkSel0,`
 - `kSimTpmClkSel1 }`

SIM Timer/PWM external clock select.
- enum `sim_tpm_ch_src_k50d10_t` {
 - `kSimTpmChSrc0,`
 - `kSimTpmChSrc1 }`

SIM Timer/PWM x channel y input capture source select.
- enum `sim_cmtuartpad_strength_k50d10_t` {
 - `kSimCmtuartSinglePad,`
 - `kSimCmtuartDualPad }`

SIM CMT/UART pad drive strength.
- enum `sim_ptd7pad_strength_k50d10_t` {
 - `kSimPtd7padSinglePad,`
 - `kSimPtd7padDualPad }`

SIM PTD7 pad drive strength.
- enum `sim_flexbus_security_level_k50d10_t` {
 - `kSimFbslLevel0,`
 - `kSimFbslLevel1,`
 - `kSimFbslLevel2,`
 - `kSimFbslLevel3 }`

SIM FlexBus security level.
- enum `sim_clock_gate_name_k50d10_t`

Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.

53.2.37.2 Macro Definition Documentation

53.2.37.2.1 `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`

53.2.37.3 Enumeration Type Documentation

53.2.37.3.1 enum `clock_wdog_src_k50d10_t`

Enumerator

- `kClockWdogSrcLpoClk` LPO.
- `kClockWdogSrcAltClk` Alternative clock, for K50D10 it is Bus clock.

53.2.37.3.2 enum `clock_trace_src_k50d10_t`

Enumerator

- `kClockTraceSrcMcgoutClk` MCG out clock.
- `kClockTraceSrcCoreClk` core clock

53.2.37.3.3 enum clock_port_filter_src_k50d10_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.*kClockPortFilterSrcLpoClk* LPO.**53.2.37.3.4 enum clock_lptmr_src_k50d10_t**

Enumerator

kClockLptmrSrcMcgIrClk MCG IRC clock.*kClockLptmrSrcLpoClk* LPO clock.*kClockLptmrSrcEr32kClk* ERCLK32K clock.*kClockLptmrSrcOsc0erClk* OSCERCLK clock.**53.2.37.3.5 enum clock_time_src_k50d10_t**

Enumerator

kClockTimeSrcCoreSysClk Core/system clock.*kClockTimeSrcPllFllSel* clock as selected by SOPT2[PLLFLLSSEL].*kClockTimeSrcOsc0erClk* OSCERCLK clock.*kClockTimeSrcExt* ENET 1588 clock in (ENET_1588_CLKIN)**53.2.37.3.6 enum clock_rmii_src_k50d10_t**

Enumerator

kClockRmiiSrcExtalClk EXTAL Clock.*kClockRmiiSrcExt* ENET 1588 clock in (ENET_1588_CLKIN)**53.2.37.3.7 enum clock_flexcan_src_k50d10_t**

Enumerator

kClockFlexcanSrcOsc0erClk OSCERCLK.*kClockFlexcanSrcBusClk* Bus clock.

SIM HAL driver

53.2.37.3.8 enum clock_sdhc_src_k50d10_t

Enumerator

- kClockSdhcSrcCoreSysClk* Core/system clock.
- kClockSdhcSrcPllFllSel* clock as selected by SOPT2[PLLFLLSEL].
- kClockSdhcSrcOsc0erClk* OSCERCLK clock.
- kClockSdhcSrcExt* External bypass clock (SDHC0_CLKIN)

53.2.37.3.9 enum clock_sai_src_k50d10_t

Enumerator

- kClockSaiSrcSysClk* SYSCLK.
- kClockSaiSrcOsc0erClk* OSC0ERCLK.
- kClockSaiSrcPllClk* MCGPLLCLK.

53.2.37.3.10 enum clock_tsi_active_mode_src_k50d10_t

Enumerator

- kClockTsiActiveSrcBusClk* Bus clock.
- kClockTsiActiveSrcMcgIrClk* MCG IRC clock.
- kClockTsiActiveSrcOsc0erClk* OSCERCLK clock.

53.2.37.3.11 enum clock_tsi_lp_mode_src_k50d10_t

Enumerator

- kClockTsiLpSrcLpoClk* LPO clock.
- kClockTsiLpSrcEr32kClk* ERCLK32K clock.

53.2.37.3.12 enum clock_pllfl_sel_k50d10_t

Enumerator

- kClockPllFllSelFll* Fll clock.
- kClockPllFllSelPll* Pll0 clock.

53.2.37.3.13 enum clock_er32k_src_k50d10_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcRtc RTC 32k clock .
kClockEr32kSrcLpo LPO clock.

53.2.37.3.14 enum clock_clkout_src_k50d10_t

Enumerator

kClockClkoutSelFlexbusClk Flexbus clock.
kClockClkoutSelFlashClk Flash clock.
kClockClkoutSelLpoClk LPO clock.
kClockClkoutSelMcgIrClk MCG out clock.
kClockClkoutSelRtc32kClk RTC 32k clock.
kClockClkoutSelOsc0erClk OSCERCLK0 clock.

53.2.37.3.15 enum clock_rtcout_src_k50d10_t

Enumerator

kClockRtcoutSrc1Hz 1Hz clock
kClockRtcoutSrc32kHz 32kHz clock

53.2.37.3.16 enum sim_adc_pretrg_sel_k50d10_t

Enumerator

kSimAdcPretrgselA Pre-trigger A selected for ADCx.
kSimAdcPretrgselB Pre-trigger B selected for ADCx.

53.2.37.3.17 enum sim_adc_trg_sel_k50d10_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.

SIM HAL driver

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.

53.2.37.3.18 enum sim_uart_rxsrc_k50d10_t

Enumerator

kSimUartRxsrcPin UARTx_RX Pin.
kSimUartRxsrcCmp0 CMP0.
kSimUartRxsrcCmp1 CMP1.

53.2.37.3.19 enum sim_uart_txsrc_k50d10_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.

53.2.37.3.20 enum sim_ftm_trg_src_k50d10_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.
kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

53.2.37.3.21 enum sim_ftm_clk_sel_k50d10_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.
kSimFtmClkSel1 FTM CLKIN1 pin.

53.2.37.3.22 enum sim_ftm_ch_src_k50d10_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.

53.2.37.3.23 enum sim_ftmflt_sel_k50d10_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

53.2.37.3.24 enum sim_tpm_clk_sel_k50d10_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM_CLKIN1 pin.

53.2.37.3.25 enum sim_tpm_ch_src_k50d10_t

Enumerator

- kSimTpmChSrc0* TPMx_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

53.2.37.3.26 enum sim_cmtuartpad_streng_h_k50d10_t

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0_TXD.

53.2.37.3.27 enum sim_ptd7pad_streng_h_k50d10_t

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

SIM HAL driver

53.2.37.3.28 enum sim_flexbus_security_level_k50d10_t

Enumerator

kSimFbslLevel0 FlexBus security level 0.

kSimFbslLevel1 FlexBus security level 1.

kSimFbslLevel2 FlexBus security level 2.

kSimFbslLevel3 FlexBus security level 3.

53.2.37.3.29 enum sim_clock_gate_name_k50d10_t

53.2.38 K51D10 SIM HAL driver

53.2.38.1 Overview

The section describes the enumerations, macros and data structures for K51D10 SIM HAL driver.

Files

- file [fsl_sim_hal_MK51D10.h](#)

Macros

- #define [FSL_SIM_SCGC_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)
SIM SCGC bit index.

Enumerations

- enum [clock_wdog_src_k51d10_t](#) {
[kClockWdogSrcLpoClk](#),
[kClockWdogSrcAltClk](#) }
WDOG clock source select.
- enum [clock_trace_src_k51d10_t](#) {
[kClockTraceSrcMcgoutClk](#),
[kClockTraceSrcCoreClk](#) }
Debug trace clock source select.
- enum [clock_port_filter_src_k51d10_t](#) {
[kClockPortFilterSrcBusClk](#),
[kClockPortFilterSrcLpoClk](#) }
PORTx digital input filter clock source select.
- enum [clock_lptmr_src_k51d10_t](#) {
[kClockLptmrSrcMcgIrClk](#),
[kClockLptmrSrcLpoClk](#),
[kClockLptmrSrcEr32kClk](#),
[kClockLptmrSrcOsc0erClk](#) }
LPTMR clock source select.
- enum [clock_time_src_k51d10_t](#) {
[kClockTimeSrcCoreSysClk](#),
[kClockTimeSrcPIIFilSel](#),
[kClockTimeSrcOsc0erClk](#),
[kClockTimeSrcExt](#) }
SIM timestamp clock source.
- enum [clock_rmii_src_k51d10_t](#) {
[kClockRmiiSrcExtalClk](#),
[kClockRmiiSrcExt](#) }
SIM RMII clock source.

SIM HAL driver

- enum `clock_flexcan_src_k51d10_t` {
`kClockFlexcanSrcOsc0erClk`,
`kClockFlexcanSrcBusClk` }
FLEXCAN clock source select.
- enum `clock_sdhc_src_k51d10_t` {
`kClockSdhcSrcCoreSysClk`,
`kClockSdhcSrcPllFllSel`,
`kClockSdhcSrcOsc0erClk`,
`kClockSdhcSrcExt` }
SDHC clock source.
- enum `clock_sai_src_k51d10_t` {
`kClockSaiSrcSysClk` = 0U,
`kClockSaiSrcOsc0erClk` = 1U,
`kClockSaiSrcPllClk` = 3U }
SAI clock source.
- enum `clock_tsi_active_mode_src_k51d10_t` {
`kClockTsiActiveSrcBusClk`,
`kClockTsiActiveSrcMcgIrClk`,
`kClockTsiActiveSrcOsc0erClk` }
TSI Active Mode clock source.
- enum `clock_tsi_lp_mode_src_k51d10_t` {
`kClockTsiLpSrcLpoClk`,
`kClockTsiLpSrcEr32kClk` }
TSI Low-power Mode clock source.
- enum `clock_pllfl_sel_k51d10_t` {
`kClockPllFllSelFll` = 0U,
`kClockPllFllSelPll` = 1U }
SIM PLLFLLSEL clock source select.
- enum `clock_er32k_src_k51d10_t` {
`kClockEr32kSrcOsc0` = 0U,
`kClockEr32kSrcRtc` = 2U,
`kClockEr32kSrcLpo` = 3U }
SIM external reference clock source select (OSC32KSEL).
- enum `clock_clkout_src_k51d10_t` {
`kClockClkoutSelFlexbusClk` = 0U,
`kClockClkoutSelFlashClk` = 2U,
`kClockClkoutSelLpoClk` = 3U,
`kClockClkoutSelMcgIrClk` = 4U,
`kClockClkoutSelRtc32kClk` = 5U,
`kClockClkoutSelOsc0erClk` = 6U }
SIM CLKOUT_SEL clock source select.
- enum `clock_rtcout_src_k51d10_t` {
`kClockRtcoutSrc1Hz`,
`kClockRtcoutSrc32kHz` }
SIM RTCCLKOUTSEL clock source select.
- enum `sim_adc_pretrg_sel_k51d10_t` {
`kSimAdcPretrgSelA`,

- kSimAdcPretrgSelB }
 - SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_k51d10_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_rxsrc_k51d10_t` {
 - kSimUartRxsrcPin,
 - kSimUartRxsrcCmp0,
 - kSimUartRxsrcCmp1 }
 - SIM UART receive data source select.*
- enum `sim_uart_txsrc_k51d10_t` {
 - kSimUartTxsrcPin,
 - kSimUartTxsrcFtm1,
 - kSimUartTxsrcFtm2 }
 - SIM UART transmit data source select.*
- enum `sim_ftm_trg_src_k51d10_t` {
 - kSimFtmTrgSrc0,
 - kSimFtmTrgSrc1 }
 - SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_k51d10_t` {
 - kSimFtmClkSel0,
 - kSimFtmClkSel1 }
 - SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_k51d10_t` {
 - kSimFtmChSrc0,
 - kSimFtmChSrc1,
 - kSimFtmChSrc2,
 - kSimFtmChSrc3 }
 - SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftmflt_sel_k51d10_t` {
 - kSimFtmFltSel0,
 - kSimFtmFltSel1 }
 - SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_k51d10_t` {

SIM HAL driver

- `kSimTpmClkSel0,`
 - `kSimTpmClkSel1 }`

SIM Timer/PWM external clock select.
- enum `sim_tpm_ch_src_k51d10_t` {
 - `kSimTpmChSrc0,`
 - `kSimTpmChSrc1 }`

SIM Timer/PWM x channel y input capture source select.
- enum `sim_cmtuartpad_strenght_k51d10_t` {
 - `kSimCmtuartSinglePad,`
 - `kSimCmtuartDualPad }`

SIM CMT/UART pad drive strength.
- enum `sim_ptd7pad_strenght_k51d10_t` {
 - `kSimPtd7padSinglePad,`
 - `kSimPtd7padDualPad }`

SIM PTD7 pad drive strength.
- enum `sim_flexbus_security_level_k51d10_t` {
 - `kSimFbslLevel0,`
 - `kSimFbslLevel1,`
 - `kSimFbslLevel2,`
 - `kSimFbslLevel3 }`

SIM FlexBus security level.
- enum `sim_clock_gate_name_k51d10_t`

Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.

53.2.38.2 Macro Definition Documentation

53.2.38.2.1 `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`

53.2.38.3 Enumeration Type Documentation

53.2.38.3.1 enum `clock_wdog_src_k51d10_t`

Enumerator

- `kClockWdogSrcLpoClk` LPO.
- `kClockWdogSrcAltClk` Alternative clock, for K51D10 it is Bus clock.

53.2.38.3.2 enum `clock_trace_src_k51d10_t`

Enumerator

- `kClockTraceSrcMcgoutClk` MCG out clock.
- `kClockTraceSrcCoreClk` core clock

53.2.38.3.3 enum clock_port_filter_src_k51d10_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.
kClockPortFilterSrcLpoClk LPO.

53.2.38.3.4 enum clock_lptmr_src_k51d10_t

Enumerator

kClockLptmrSrcMcgIrClk MCG IRC clock.
kClockLptmrSrcLpoClk LPO clock.
kClockLptmrSrcEr32kClk ERCLK32K clock.
kClockLptmrSrcOsc0erClk OSCERCLK clock.

53.2.38.3.5 enum clock_time_src_k51d10_t

Enumerator

kClockTimeSrcCoreSysClk Core/system clock.
kClockTimeSrcPllFllSel clock as selected by SOPT2[PLLFLLSSEL].
kClockTimeSrcOsc0erClk OSCERCLK clock.
kClockTimeSrcExt ENET 1588 clock in (ENET_1588_CLKIN)

53.2.38.3.6 enum clock_rmii_src_k51d10_t

Enumerator

kClockRmiiSrcExtalClk EXTAL Clock.
kClockRmiiSrcExt ENET 1588 clock in (ENET_1588_CLKIN)

53.2.38.3.7 enum clock_flexcan_src_k51d10_t

Enumerator

kClockFlexcanSrcOsc0erClk OSCERCLK.
kClockFlexcanSrcBusClk Bus clock.

SIM HAL driver

53.2.38.3.8 enum clock_sdhc_src_k51d10_t

Enumerator

kClockSdhcSrcCoreSysClk Core/system clock.
kClockSdhcSrcPllFllSel clock as selected by SOPT2[PLLFLLSEL].
kClockSdhcSrcOsc0erClk OSCERCLK clock.
kClockSdhcSrcExt External bypass clock (SDHC0_CLKIN)

53.2.38.3.9 enum clock_sai_src_k51d10_t

Enumerator

kClockSaiSrcSysClk SYSCLK.
kClockSaiSrcOsc0erClk OSC0ERCLK.
kClockSaiSrcPllClk MCGPLLCLK.

53.2.38.3.10 enum clock_tsi_active_mode_src_k51d10_t

Enumerator

kClockTsiActiveSrcBusClk Bus clock.
kClockTsiActiveSrcMcgIrClk MCG IRC clock.
kClockTsiActiveSrcOsc0erClk OSCERCLK clock.

53.2.38.3.11 enum clock_tsi_lp_mode_src_k51d10_t

Enumerator

kClockTsiLpSrcLpoClk LPO clock.
kClockTsiLpSrcEr32kClk ERCLK32K clock.

53.2.38.3.12 enum clock_pllfl_sel_k51d10_t

Enumerator

kClockPllFllSelFll Fll clock.
kClockPllFllSelPll Pll0 clock.

53.2.38.3.13 enum clock_er32k_src_k51d10_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcRtc RTC 32k clock .
kClockEr32kSrcLpo LPO clock.

53.2.38.3.14 enum clock_clkout_src_k51d10_t

Enumerator

kClockClkoutSelFlexbusClk Flexbus clock.
kClockClkoutSelFlashClk Flash clock.
kClockClkoutSelLpoClk LPO clock.
kClockClkoutSelMcgIrClk MCG out clock.
kClockClkoutSelRtc32kClk RTC 32k clock.
kClockClkoutSelOsc0erClk OSCERCLK0 clock.

53.2.38.3.15 enum clock_rtcout_src_k51d10_t

Enumerator

kClockRtcoutSrc1Hz 1Hz clock
kClockRtcoutSrc32kHz 32kHz clock

53.2.38.3.16 enum sim_adc_pretrg_sel_k51d10_t

Enumerator

kSimAdcPretrgSelA Pre-trigger A selected for ADCx.
kSimAdcPretrgSelB Pre-trigger B selected for ADCx.

53.2.38.3.17 enum sim_adc_trg_sel_k51d10_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.

SIM HAL driver

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.

53.2.38.3.18 enum sim_uart_rxsrc_k51d10_t

Enumerator

kSimUartRxsrcPin UARTx_RX Pin.
kSimUartRxsrcCmp0 CMP0.
kSimUartRxsrcCmp1 CMP1.

53.2.38.3.19 enum sim_uart_txsrc_k51d10_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.

53.2.38.3.20 enum sim_ftm_trg_src_k51d10_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.
kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

53.2.38.3.21 enum sim_ftm_clk_sel_k51d10_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.
kSimFtmClkSel1 FTM CLKIN1 pin.

53.2.38.3.22 enum sim_ftm_ch_src_k51d10_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.

53.2.38.3.23 enum sim_ftmflt_sel_k51d10_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

53.2.38.3.24 enum sim_tpm_clk_sel_k51d10_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM_CLKIN1 pin.

53.2.38.3.25 enum sim_tpm_ch_src_k51d10_t

Enumerator

- kSimTpmChSrc0* TPMx_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

53.2.38.3.26 enum sim_cmtuartpad_streng_h_k51d10_t

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0_TXD.

53.2.38.3.27 enum sim_ptd7pad_streng_h_k51d10_t

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

SIM HAL driver

53.2.38.3.28 enum sim_flexbus_security_level_k51d10_t

Enumerator

kSimFbslLevel0 FlexBus security level 0.

kSimFbslLevel1 FlexBus security level 1.

kSimFbslLevel2 FlexBus security level 2.

kSimFbslLevel3 FlexBus security level 3.

53.2.38.3.29 enum sim_clock_gate_name_k51d10_t

53.2.39 K52D10 SIM HAL driver

53.2.39.1 Overview

The section describes the enumerations, macros and data structures for K52D10 SIM HAL driver.

Files

- file [fsl_sim_hal_MK52D10.h](#)

Macros

- #define [FSL_SIM_SCGC_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)
SIM SCGC bit index.

Enumerations

- enum [clock_wdog_src_k52d10_t](#) {
[kClockWdogSrcLpoClk](#),
[kClockWdogSrcAltClk](#) }
WDOG clock source select.
- enum [clock_trace_src_k52d10_t](#) {
[kClockTraceSrcMcgoutClk](#),
[kClockTraceSrcCoreClk](#) }
Debug trace clock source select.
- enum [clock_port_filter_src_k52d10_t](#) {
[kClockPortFilterSrcBusClk](#),
[kClockPortFilterSrcLpoClk](#) }
PORTx digital input filter clock source select.
- enum [clock_lptmr_src_k52d10_t](#) {
[kClockLptmrSrcMcgIrClk](#),
[kClockLptmrSrcLpoClk](#),
[kClockLptmrSrcEr32kClk](#),
[kClockLptmrSrcOsc0erClk](#) }
LPTMR clock source select.
- enum [clock_time_src_k52d10_t](#) {
[kClockTimeSrcCoreSysClk](#),
[kClockTimeSrcPIIFilSel](#),
[kClockTimeSrcOsc0erClk](#),
[kClockTimeSrcExt](#) }
SIM timestamp clock source.
- enum [clock_rmii_src_k52d10_t](#) {
[kClockRmiiSrcExtalClk](#),
[kClockRmiiSrcExt](#) }
SIM RMII clock source.

SIM HAL driver

- enum `clock_flexcan_src_k52d10_t` {
`kClockFlexcanSrcOsc0erClk`,
`kClockFlexcanSrcBusClk` }
FLEXCAN clock source select.
- enum `clock_sdhc_src_k52d10_t` {
`kClockSdhcSrcCoreSysClk`,
`kClockSdhcSrcPllFllSel`,
`kClockSdhcSrcOsc0erClk`,
`kClockSdhcSrcExt` }
SDHC clock source.
- enum `clock_sai_src_k52d10_t` {
`kClockSaiSrcSysClk` = 0U,
`kClockSaiSrcOsc0erClk` = 1U,
`kClockSaiSrcPllClk` = 3U }
SAI clock source.
- enum `clock_tsi_active_mode_src_k52d10_t` {
`kClockTsiActiveSrcBusClk`,
`kClockTsiActiveSrcMcgIrClk`,
`kClockTsiActiveSrcOsc0erClk` }
TSI Active Mode clock source.
- enum `clock_tsi_lp_mode_src_k52d10_t` {
`kClockTsiLpSrcLpoClk`,
`kClockTsiLpSrcEr32kClk` }
TSI Low-power Mode clock source.
- enum `clock_pllfl_sel_k52d10_t` {
`kClockPllFllSelFll` = 0U,
`kClockPllFllSelPll` = 1U }
SIM PLLFLLSEL clock source select.
- enum `clock_er32k_src_k52d10_t` {
`kClockEr32kSrcOsc0` = 0U,
`kClockEr32kSrcRtc` = 2U,
`kClockEr32kSrcLpo` = 3U }
SIM external reference clock source select (OSC32KSEL).
- enum `clock_clkout_src_k52d10_t` {
`kClockClkoutSelFlexbusClk` = 0U,
`kClockClkoutSelFlashClk` = 2U,
`kClockClkoutSelLpoClk` = 3U,
`kClockClkoutSelMcgIrClk` = 4U,
`kClockClkoutSelRtc32kClk` = 5U,
`kClockClkoutSelOsc0erClk` = 6U }
SIM CLKOUT_SEL clock source select.
- enum `clock_rtcout_src_k52d10_t` {
`kClockRtcoutSrc1Hz`,
`kClockRtcoutSrc32kHz` }
SIM RTCCLKOUTSEL clock source select.
- enum `sim_adc_pretrg_sel_k52d10_t` {
`kSimAdcPretrgSelA`,

- kSimAdcPretrgSelB }
 - SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_k52d10_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_rxsrc_k52d10_t` {
 - kSimUartRxsrcPin,
 - kSimUartRxsrcCmp0,
 - kSimUartRxsrcCmp1 }
 - SIM UART receive data source select.*
- enum `sim_uart_txsrc_k52d10_t` {
 - kSimUartTxsrcPin,
 - kSimUartTxsrcFtm1,
 - kSimUartTxsrcFtm2 }
 - SIM UART transmit data source select.*
- enum `sim_ftm_trg_src_k52d10_t` {
 - kSimFtmTrgSrc0,
 - kSimFtmTrgSrc1 }
 - SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_k52d10_t` {
 - kSimFtmClkSel0,
 - kSimFtmClkSel1 }
 - SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_k52d10_t` {
 - kSimFtmChSrc0,
 - kSimFtmChSrc1,
 - kSimFtmChSrc2,
 - kSimFtmChSrc3 }
 - SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftmflt_sel_k52d10_t` {
 - kSimFtmFltSel0,
 - kSimFtmFltSel1 }
 - SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_k52d10_t` {

SIM HAL driver

- `kSimTpmClkSel0,`
 - `kSimTpmClkSel1 }`

SIM Timer/PWM external clock select.
- enum `sim_tpm_ch_src_k52d10_t` {
 - `kSimTpmChSrc0,`
 - `kSimTpmChSrc1 }`

SIM Timer/PWM x channel y input capture source select.
- enum `sim_cmtuartpad_strength_k52d10_t` {
 - `kSimCmtuartSinglePad,`
 - `kSimCmtuartDualPad }`

SIM CMT/UART pad drive strength.
- enum `sim_ptd7pad_strength_k52d10_t` {
 - `kSimPtd7padSinglePad,`
 - `kSimPtd7padDualPad }`

SIM PTD7 pad drive strength.
- enum `sim_flexbus_security_level_k52d10_t` {
 - `kSimFbslLevel0,`
 - `kSimFbslLevel1,`
 - `kSimFbslLevel2,`
 - `kSimFbslLevel3 }`

SIM FlexBus security level.
- enum `sim_clock_gate_name_k52d10_t`

Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.

53.2.39.2 Macro Definition Documentation

53.2.39.2.1 `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`

53.2.39.3 Enumeration Type Documentation

53.2.39.3.1 enum `clock_wdog_src_k52d10_t`

Enumerator

- `kClockWdogSrcLpoClk` LPO.
- `kClockWdogSrcAltClk` Alternative clock, for K52D10 it is Bus clock.

53.2.39.3.2 enum `clock_trace_src_k52d10_t`

Enumerator

- `kClockTraceSrcMcgoutClk` MCG out clock.
- `kClockTraceSrcCoreClk` core clock

53.2.39.3.3 enum clock_port_filter_src_k52d10_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.*kClockPortFilterSrcLpoClk* LPO.**53.2.39.3.4 enum clock_lptmr_src_k52d10_t**

Enumerator

kClockLptmrSrcMcgIrClk MCG IRC clock.*kClockLptmrSrcLpoClk* LPO clock.*kClockLptmrSrcEr32kClk* ERCLK32K clock.*kClockLptmrSrcOsc0erClk* OSCERCLK clock.**53.2.39.3.5 enum clock_time_src_k52d10_t**

Enumerator

kClockTimeSrcCoreSysClk Core/system clock.*kClockTimeSrcPllFllSel* clock as selected by SOPT2[PLLFLLSSEL].*kClockTimeSrcOsc0erClk* OSCERCLK clock.*kClockTimeSrcExt* ENET 1588 clock in (ENET_1588_CLKIN)**53.2.39.3.6 enum clock_rmii_src_k52d10_t**

Enumerator

kClockRmiiSrcExtalClk EXTAL Clock.*kClockRmiiSrcExt* ENET 1588 clock in (ENET_1588_CLKIN)**53.2.39.3.7 enum clock_flexcan_src_k52d10_t**

Enumerator

kClockFlexcanSrcOsc0erClk OSCERCLK.*kClockFlexcanSrcBusClk* Bus clock.

SIM HAL driver

53.2.39.3.8 enum clock_sdhc_src_k52d10_t

Enumerator

kClockSdhcSrcCoreSysClk Core/system clock.
kClockSdhcSrcPllFllSel clock as selected by SOPT2[PLLFLLSEL].
kClockSdhcSrcOsc0erClk OSCERCLK clock.
kClockSdhcSrcExt External bypass clock (SDHC0_CLKIN)

53.2.39.3.9 enum clock_sai_src_k52d10_t

Enumerator

kClockSaiSrcSysClk SYSCLK.
kClockSaiSrcOsc0erClk OSC0ERCLK.
kClockSaiSrcPllClk MCGPLLCLK.

53.2.39.3.10 enum clock_tsi_active_mode_src_k52d10_t

Enumerator

kClockTsiActiveSrcBusClk Bus clock.
kClockTsiActiveSrcMcgIrClk MCG IRC clock.
kClockTsiActiveSrcOsc0erClk OSCERCLK clock.

53.2.39.3.11 enum clock_tsi_lp_mode_src_k52d10_t

Enumerator

kClockTsiLpSrcLpoClk LPO clock.
kClockTsiLpSrcEr32kClk ERCLK32K clock.

53.2.39.3.12 enum clock_pllfl_sel_k52d10_t

Enumerator

kClockPllFllSelFll Fll clock.
kClockPllFllSelPll Pll0 clock.

53.2.39.3.13 enum clock_er32k_src_k52d10_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcRtc RTC 32k clock .
kClockEr32kSrcLpo LPO clock.

53.2.39.3.14 enum clock_clkout_src_k52d10_t

Enumerator

kClockClkoutSelFlexbusClk Flexbus clock.
kClockClkoutSelFlashClk Flash clock.
kClockClkoutSelLpoClk LPO clock.
kClockClkoutSelMcgIrClk MCG out clock.
kClockClkoutSelRtc32kClk RTC 32k clock.
kClockClkoutSelOsc0erClk OSCERCLK0 clock.

53.2.39.3.15 enum clock_rtcout_src_k52d10_t

Enumerator

kClockRtcoutSrc1Hz 1Hz clock
kClockRtcoutSrc32kHz 32kHz clock

53.2.39.3.16 enum sim_adc_pretrg_sel_k52d10_t

Enumerator

kSimAdcPretrgSelA Pre-trigger A selected for ADCx.
kSimAdcPretrgSelB Pre-trigger B selected for ADCx.

53.2.39.3.17 enum sim_adc_trg_sel_k52d10_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.

SIM HAL driver

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.

53.2.39.3.18 enum sim_uart_rxsrc_k52d10_t

Enumerator

kSimUartRxsrcPin UARTx_RX Pin.
kSimUartRxsrcCmp0 CMP0.
kSimUartRxsrcCmp1 CMP1.

53.2.39.3.19 enum sim_uart_txsrc_k52d10_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.

53.2.39.3.20 enum sim_ftm_trg_src_k52d10_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.
kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

53.2.39.3.21 enum sim_ftm_clk_sel_k52d10_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.
kSimFtmClkSel1 FTM CLKIN1 pin.

53.2.39.3.22 enum sim_ftm_ch_src_k52d10_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.

53.2.39.3.23 enum sim_ftmflt_sel_k52d10_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

53.2.39.3.24 enum sim_tpm_clk_sel_k52d10_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM_CLKIN1 pin.

53.2.39.3.25 enum sim_tpm_ch_src_k52d10_t

Enumerator

- kSimTpmChSrc0* TPMx_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

53.2.39.3.26 enum sim_cmtuartpad_streng_h_k52d10_t

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0_TXD.

53.2.39.3.27 enum sim_ptd7pad_streng_h_k52d10_t

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

SIM HAL driver

53.2.39.3.28 enum sim_flexbus_security_level_k52d10_t

Enumerator

kSimFbslLevel0 FlexBus security level 0.

kSimFbslLevel1 FlexBus security level 1.

kSimFbslLevel2 FlexBus security level 2.

kSimFbslLevel3 FlexBus security level 3.

53.2.39.3.29 enum sim_clock_gate_name_k52d10_t

53.2.40 K53D10 SIM HAL driver

53.2.40.1 Overview

The section describes the enumerations, macros and data structures for K53D10 SIM HAL driver.

Files

- file [fsl_sim_hal_MK53D10.h](#)

Macros

- #define [FSL_SIM_SCGC_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)
SIM SCGC bit index.

Enumerations

- enum [clock_wdog_src_k53d10_t](#) {
[kClockWdogSrcLpoClk](#),
[kClockWdogSrcAltClk](#) }
WDOG clock source select.
- enum [clock_trace_src_k53d10_t](#) {
[kClockTraceSrcMcgoutClk](#),
[kClockTraceSrcCoreClk](#) }
Debug trace clock source select.
- enum [clock_port_filter_src_k53d10_t](#) {
[kClockPortFilterSrcBusClk](#),
[kClockPortFilterSrcLpoClk](#) }
PORTx digital input filter clock source select.
- enum [clock_lptmr_src_k53d10_t](#) {
[kClockLptmrSrcMcgIrClk](#),
[kClockLptmrSrcLpoClk](#),
[kClockLptmrSrcEr32kClk](#),
[kClockLptmrSrcOsc0erClk](#) }
LPTMR clock source select.
- enum [clock_time_src_k53d10_t](#) {
[kClockTimeSrcCoreSysClk](#),
[kClockTimeSrcPIIFilSel](#),
[kClockTimeSrcOsc0erClk](#),
[kClockTimeSrcExt](#) }
SIM timestamp clock source.
- enum [clock_rmii_src_k53d10_t](#) {
[kClockRmiiSrcExtalClk](#),
[kClockRmiiSrcExt](#) }
SIM RMII clock source.

SIM HAL driver

- enum `clock_flexcan_src_k53d10_t` {
`kClockFlexcanSrcOsc0erClk`,
`kClockFlexcanSrcBusClk` }
FLEXCAN clock source select.
- enum `clock_sdhc_src_k53d10_t` {
`kClockSdhcSrcCoreSysClk`,
`kClockSdhcSrcPllFllSel`,
`kClockSdhcSrcOsc0erClk`,
`kClockSdhcSrcExt` }
SDHC clock source.
- enum `clock_sai_src_k53d10_t` {
`kClockSaiSrcSysClk` = 0U,
`kClockSaiSrcOsc0erClk` = 1U,
`kClockSaiSrcPllClk` = 3U }
SAI clock source.
- enum `clock_tsi_active_mode_src_k53d10_t` {
`kClockTsiActiveSrcBusClk`,
`kClockTsiActiveSrcMcgIrClk`,
`kClockTsiActiveSrcOsc0erClk` }
TSI Active Mode clock source.
- enum `clock_tsi_lp_mode_src_k53d10_t` {
`kClockTsiLpSrcLpoClk`,
`kClockTsiLpSrcEr32kClk` }
TSI Low-power Mode clock source.
- enum `clock_pllfl_sel_k53d10_t` {
`kClockPllFllSelFll` = 0U,
`kClockPllFllSelPll` = 1U }
SIM PLLFLLSEL clock source select.
- enum `clock_er32k_src_k53d10_t` {
`kClockEr32kSrcOsc0` = 0U,
`kClockEr32kSrcRtc` = 2U,
`kClockEr32kSrcLpo` = 3U }
SIM external reference clock source select (OSC32KSEL).
- enum `clock_clkout_src_k53d10_t` {
`kClockClkoutSelFlexbusClk` = 0U,
`kClockClkoutSelFlashClk` = 2U,
`kClockClkoutSelLpoClk` = 3U,
`kClockClkoutSelMcgIrClk` = 4U,
`kClockClkoutSelRtc32kClk` = 5U,
`kClockClkoutSelOsc0erClk` = 6U }
SIM CLKOUT_SEL clock source select.
- enum `clock_rtcout_src_k53d10_t` {
`kClockRtcoutSrc1Hz`,
`kClockRtcoutSrc32kHz` }
SIM RTCCLKOUTSEL clock source select.
- enum `sim_adc_pretrg_sel_k53d10_t` {
`kSimAdcPretrgselA`,

- `kSimAdcPretrgSelB }`
 - SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_k53d10_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_rxsrc_k53d10_t` {
 - `kSimUartRxsrcPin,`
 - `kSimUartRxsrcCmp0,`
 - `kSimUartRxsrcCmp1 }`
 - SIM UART receive data source select.*
- enum `sim_uart_txsrc_k53d10_t` {
 - `kSimUartTxsrcPin,`
 - `kSimUartTxsrcFtm1,`
 - `kSimUartTxsrcFtm2 }`
 - SIM UART transmit data source select.*
- enum `sim_ftm_trg_src_k53d10_t` {
 - `kSimFtmTrgSrc0,`
 - `kSimFtmTrgSrc1 }`
 - SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_k53d10_t` {
 - `kSimFtmClkSel0,`
 - `kSimFtmClkSel1 }`
 - SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_k53d10_t` {
 - `kSimFtmChSrc0,`
 - `kSimFtmChSrc1,`
 - `kSimFtmChSrc2,`
 - `kSimFtmChSrc3 }`
 - SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftmflt_sel_k53d10_t` {
 - `kSimFtmFltSel0,`
 - `kSimFtmFltSel1 }`
 - SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_k53d10_t` {

SIM HAL driver

- `kSimTpmClkSel0,`
 - `kSimTpmClkSel1 }`

SIM Timer/PWM external clock select.
- enum `sim_tpm_ch_src_k53d10_t` {
 - `kSimTpmChSrc0,`
 - `kSimTpmChSrc1 }`

SIM Timer/PWM x channel y input capture source select.
- enum `sim_cmtuartpad_strength_k53d10_t` {
 - `kSimCmtuartSinglePad,`
 - `kSimCmtuartDualPad }`

SIM CMT/UART pad drive strength.
- enum `sim_ptd7pad_strength_k53d10_t` {
 - `kSimPtd7padSinglePad,`
 - `kSimPtd7padDualPad }`

SIM PTD7 pad drive strength.
- enum `sim_flexbus_security_level_k53d10_t` {
 - `kSimFbslLevel0,`
 - `kSimFbslLevel1,`
 - `kSimFbslLevel2,`
 - `kSimFbslLevel3 }`

SIM FlexBus security level.
- enum `sim_clock_gate_name_k53d10_t`

Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.

53.2.40.2 Macro Definition Documentation

53.2.40.2.1 `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`

53.2.40.3 Enumeration Type Documentation

53.2.40.3.1 enum `clock_wdog_src_k53d10_t`

Enumerator

- `kClockWdogSrcLpoClk` LPO.
- `kClockWdogSrcAltClk` Alternative clock, for K53D10 it is Bus clock.

53.2.40.3.2 enum `clock_trace_src_k53d10_t`

Enumerator

- `kClockTraceSrcMcgoutClk` MCG out clock.
- `kClockTraceSrcCoreClk` core clock

53.2.40.3.3 enum clock_port_filter_src_k53d10_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.
kClockPortFilterSrcLpoClk LPO.

53.2.40.3.4 enum clock_lptmr_src_k53d10_t

Enumerator

kClockLptmrSrcMcgIrClk MCG IRC clock.
kClockLptmrSrcLpoClk LPO clock.
kClockLptmrSrcEr32kClk ERCLK32K clock.
kClockLptmrSrcOsc0erClk OSCERCLK clock.

53.2.40.3.5 enum clock_time_src_k53d10_t

Enumerator

kClockTimeSrcCoreSysClk Core/system clock.
kClockTimeSrcPllFllSel clock as selected by SOPT2[PLLFLLSSEL].
kClockTimeSrcOsc0erClk OSCERCLK clock.
kClockTimeSrcExt ENET 1588 clock in (ENET_1588_CLKIN)

53.2.40.3.6 enum clock_rmii_src_k53d10_t

Enumerator

kClockRmiiSrcExtalClk EXTAL Clock.
kClockRmiiSrcExt ENET 1588 clock in (ENET_1588_CLKIN)

53.2.40.3.7 enum clock_flexcan_src_k53d10_t

Enumerator

kClockFlexcanSrcOsc0erClk OSCERCLK.
kClockFlexcanSrcBusClk Bus clock.

SIM HAL driver

53.2.40.3.8 enum clock_sdhc_src_k53d10_t

Enumerator

- kClockSdhcSrcCoreSysClk* Core/system clock.
- kClockSdhcSrcPllFllSel* clock as selected by SOPT2[PLLFLLSEL].
- kClockSdhcSrcOsc0erClk* OSCERCLK clock.
- kClockSdhcSrcExt* External bypass clock (SDHC0_CLKIN)

53.2.40.3.9 enum clock_sai_src_k53d10_t

Enumerator

- kClockSaiSrcSysClk* SYSCLK.
- kClockSaiSrcOsc0erClk* OSC0ERCLK.
- kClockSaiSrcPllClk* MCGPLLCLK.

53.2.40.3.10 enum clock_tsi_active_mode_src_k53d10_t

Enumerator

- kClockTsiActiveSrcBusClk* Bus clock.
- kClockTsiActiveSrcMcgIrClk* MCG IRC clock.
- kClockTsiActiveSrcOsc0erClk* OSCERCLK clock.

53.2.40.3.11 enum clock_tsi_lp_mode_src_k53d10_t

Enumerator

- kClockTsiLpSrcLpoClk* LPO clock.
- kClockTsiLpSrcEr32kClk* ERCLK32K clock.

53.2.40.3.12 enum clock_pllfl_sel_k53d10_t

Enumerator

- kClockPllFllSelFll* Fll clock.
- kClockPllFllSelPl10* Pl10 clock.

53.2.40.3.13 enum clock_er32k_src_k53d10_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcRtc RTC 32k clock .
kClockEr32kSrcLpo LPO clock.

53.2.40.3.14 enum clock_clkout_src_k53d10_t

Enumerator

kClockClkoutSelFlexbusClk Flexbus clock.
kClockClkoutSelFlashClk Flash clock.
kClockClkoutSelLpoClk LPO clock.
kClockClkoutSelMcgIrClk MCG out clock.
kClockClkoutSelRtc32kClk RTC 32k clock.
kClockClkoutSelOsc0erClk OSCERCLK0 clock.

53.2.40.3.15 enum clock_rtcout_src_k53d10_t

Enumerator

kClockRtcoutSrc1Hz 1Hz clock
kClockRtcoutSrc32kHz 32kHz clock

53.2.40.3.16 enum sim_adc_pretrg_sel_k53d10_t

Enumerator

kSimAdcPretrgselA Pre-trigger A selected for ADCx.
kSimAdcPretrgselB Pre-trigger B selected for ADCx.

53.2.40.3.17 enum sim_adc_trg_sel_k53d10_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.

SIM HAL driver

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.

53.2.40.3.18 enum sim_uart_rxsrc_k53d10_t

Enumerator

kSimUartRxsrcPin UARTx_RX Pin.
kSimUartRxsrcCmp0 CMP0.
kSimUartRxsrcCmp1 CMP1.

53.2.40.3.19 enum sim_uart_txsrc_k53d10_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.

53.2.40.3.20 enum sim_ftm_trg_src_k53d10_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.
kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

53.2.40.3.21 enum sim_ftm_clk_sel_k53d10_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.
kSimFtmClkSel1 FTM CLKIN1 pin.

53.2.40.3.22 enum sim_ftm_ch_src_k53d10_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.

53.2.40.3.23 enum sim_ftmflt_sel_k53d10_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

53.2.40.3.24 enum sim_tpm_clk_sel_k53d10_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM_CLKIN1 pin.

53.2.40.3.25 enum sim_tpm_ch_src_k53d10_t

Enumerator

- kSimTpmChSrc0* TPMx_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

53.2.40.3.26 enum sim_cmtuartpad_streng_h_k53d10_t

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0_TXD.

53.2.40.3.27 enum sim_ptd7pad_streng_h_k53d10_t

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

SIM HAL driver

53.2.40.3.28 enum sim_flexbus_security_level_k53d10_t

Enumerator

kSimFbslLevel0 FlexBus security level 0.

kSimFbslLevel1 FlexBus security level 1.

kSimFbslLevel2 FlexBus security level 2.

kSimFbslLevel3 FlexBus security level 3.

53.2.40.3.29 enum sim_clock_gate_name_k53d10_t

53.2.41 KV10Z7 SIM HAL driver

53.2.41.1 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_rxsrc_kv10z7_t` {
 - kSimUartRxsrcPin,
 - kSimUartRxsrcCmp0,
 - kSimUartRxsrcCmp1 }
 - SIM UART receive data source select.*
 - enum `sim_uart_txsrc_kv10z7_t` {
 - kSimUartTxsrcPin,
 - kSimUartTxsrcFtm1,
 - kSimUartTxsrcFtm2 }
 - SIM UART 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_ft_sel_kv10z7_t` {
`kSimFtmFltSel0`,
`kSimFtmFltSel1` }
- SIM FlexTimer x Fault y select.*

 - enum `sim_ftm_ft_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.

53.2.41.2 Macro Definition Documentation

53.2.41.2.1 `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`

53.2.41.3 Enumeration Type Documentation

53.2.41.3.1 enum `clock_wdog_src_kv10z7_t`

Enumerator

kClockWdogSrcLpoClk LPO.
kClockWdogSrcAltClk Alternative clock.

53.2.41.3.2 enum `clock_lptmr_src_kv10z7_t`

Enumerator

kClockLptmrSrcMcgIrClk MCG IRC clock.
kClockLptmrSrcLpoClk LPO clock.
kClockLptmrSrcEr32kClk ERCLK32K clock.
kClockLptmrSrcOsc0erClk OSCERCLK clock.

53.2.41.3.3 enum `clock_er32k_src_kv10z7_t`

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcLpo LPO clock.

SIM HAL driver

53.2.41.3.4 enum clock_clkout_src_kv10z7_t

Enumerator

- kClockClkoutSelBusClk* Bus clock.
- kClockClkoutSelLpoClk* LPO clock.
- kClockClkoutSelMcgIrClk* MCG IRC clock.
- kClockClkoutSelOsc0erClk* OSCERCLK0 clock.

53.2.41.3.5 enum sim_adc_pretrg_sel_kv10z7_t

Enumerator

- kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.
- kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

53.2.41.3.6 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.

53.2.41.3.7 enum sim_uart_rxsrc_kv10z7_t

Enumerator

- kSimUartRxsrcPin* UARTx_RX Pin.
- kSimUartRxsrcCmp0* CMP0.
- kSimUartRxsrcCmp1* CMP1.

53.2.41.3.8 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.

53.2.41.3.9 enum sim_ftm_trg_src_kv10z7_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.

kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

53.2.41.3.10 enum sim_ftm_clk_sel_kv10z7_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.

kSimFtmClkSel1 FTM CLKIN1 pin.

kSimFtmClkSel2 FTM CLKIN2 pin.

53.2.41.3.11 enum clock_ftm_fixedfreq_src_kv10z7_t

Enumerator

kClockFtmClkMcgFfClk MCGFFCLK.

kClockFtmClkMcgIrClk MCGIRCLK.

kClockFtmClkOsc0erClk OSCERCLK.

53.2.41.3.12 enum clock_adc_alt_src_kv10z7_t

Enumerator

kClockAdcAltClkSrcOutdiv5 OUTDIV5 output clock.

kClockAdcAltClkSrcMcgIrClk MCGIRCLK.

kClockAdcAltClkSrcOsc0erClk OSCERCLK.

SIM HAL driver

53.2.41.3.13 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.

53.2.41.3.14 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.

53.2.41.3.15 enum sim_ftmflt_sel_kv10z7_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

53.2.41.3.16 enum sim_ftmflt_carrier_sel_kv10z7_t

Enumerator

- kSimFtmCarrierSel0* Carrier frequency selection 0.
- kSimFtmCarrierSel1* Carrier frequency selection 1.

53.2.41.3.17 enum sim_clock_gate_name_kv10z7_t

53.2.42 K60D10 SIM HAL driver

53.2.42.1 Overview

The section describes the enumerations, macros and data structures for K60D10 SIM HAL driver. K60D10 SIM code is shared by K10D10, K20D10, K30D10, K40D10, K50D10, K51D10, K52D10, K53D10 and K60D10.

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](#),
 [kClockTimeSrcPIIFllSel](#),
 [kClockTimeSrcOsc0erClk](#),
 [kClockTimeSrcExt](#) }
 SIM timestamp clock source.
- enum [clock_rmii_src_k60d10_t](#) {
 [kClockRmiiSrcExtalClk](#),

SIM HAL driver

- `kClockRmiiSrcExt` }
 - SIM RMII clock source.*
- enum `clock_flexcan_src_k60d10_t` {
 - `kClockFlexcanSrcOsc0erClk`,
 - `kClockFlexcanSrcBusClk` }
 - FLEXCAN clock source select.*
- enum `clock_sdhc_src_k60d10_t` {
 - `kClockSdhcSrcCoreSysClk`,
 - `kClockSdhcSrcPIIFIISel`,
 - `kClockSdhcSrcOsc0erClk`,
 - `kClockSdhcSrcExt` }
 - SDHC clock source.*
- enum `clock_sai_src_k60d10_t` {
 - `kClockSaiSrcSysClk = 0U`,
 - `kClockSaiSrcOsc0erClk = 1U`,
 - `kClockSaiSrcPIIClk = 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` {
 - `kClockPIIFIISeIFII = 0U`,
 - `kClockPIIFIISePII = 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_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_rxsrc_k60d10_t` {
`kSimUartRxsrcPin`,
`kSimUartRxsrcCmp0`,
`kSimUartRxsrcCmp1` }
SIM UART receive data source select.
- enum `sim_uart_txsrc_k60d10_t` {
`kSimUartTxsrcPin`,
`kSimUartTxsrcFtm1`,
`kSimUartTxsrcFtm2` }
SIM UART 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 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_ftmflt_sel_k60d10_t` {
`kSimFtmFltSel0`,

SIM HAL driver

- `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_h_k60d10_t` {
`kSimCmtuartSinglePad`,
`kSimCmtuartDualPad` }
SIM CMT/UART pad drive strength.
- enum `sim_ptd7pad_streng_h_k60d10_t` {
`kSimPtd7padSinglePad`,
`kSimPtd7padDualPad` }
SIM PTD7 pad drive strength.
- enum `sim_flexbus_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.

53.2.42.2 Macro Definition Documentation

53.2.42.2.1 `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`

53.2.42.3 Enumeration Type Documentation

53.2.42.3.1 enum `clock_wdog_src_k60d10_t`

Enumerator

`kClockWdogSrcLpoClk` LPO.

`kClockWdogSrcAltClk` Alternative clock, for K60D10 it is Bus clock.

53.2.42.3.2 enum `clock_trace_src_k60d10_t`

Enumerator

`kClockTraceSrcMcgoutClk` MCG out clock.

`kClockTraceSrcCoreClk` core clock

53.2.42.3.3 enum clock_port_filter_src_k60d10_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.
kClockPortFilterSrcLpoClk LPO.

53.2.42.3.4 enum clock_lptmr_src_k60d10_t

Enumerator

kClockLptmrSrcMcgIrClk MCG IRC clock.
kClockLptmrSrcLpoClk LPO clock.
kClockLptmrSrcEr32kClk ERCLK32K clock.
kClockLptmrSrcOsc0erClk OSCERCLK clock.

53.2.42.3.5 enum clock_time_src_k60d10_t

Enumerator

kClockTimeSrcCoreSysClk Core/system clock.
kClockTimeSrcPllFllSel clock as selected by SOPT2[PLLFLLSSEL].
kClockTimeSrcOsc0erClk OSCERCLK clock.
kClockTimeSrcExt ENET 1588 clock in (ENET_1588_CLKIN)

53.2.42.3.6 enum clock_rmii_src_k60d10_t

Enumerator

kClockRmiiSrcExtalClk EXTAL Clock.
kClockRmiiSrcExt ENET 1588 clock in (ENET_1588_CLKIN)

53.2.42.3.7 enum clock_flexcan_src_k60d10_t

Enumerator

kClockFlexcanSrcOsc0erClk OSCERCLK.
kClockFlexcanSrcBusClk Bus clock.

SIM HAL driver

53.2.42.3.8 enum clock_sdhc_src_k60d10_t

Enumerator

kClockSdhcSrcCoreSysClk Core/system clock.
kClockSdhcSrcPllFllSel clock as selected by SOPT2[PLLFLLSEL].
kClockSdhcSrcOsc0erClk OSCERCLK clock.
kClockSdhcSrcExt External bypass clock (SDHC0_CLKIN)

53.2.42.3.9 enum clock_sai_src_k60d10_t

Enumerator

kClockSaiSrcSysClk SYSCLK.
kClockSaiSrcOsc0erClk OSC0ERCLK.
kClockSaiSrcPllClk MCGPLLCLK.

53.2.42.3.10 enum clock_tsi_active_mode_src_k60d10_t

Enumerator

kClockTsiActiveSrcBusClk Bus clock.
kClockTsiActiveSrcMcgIrClk MCG IRC clock.
kClockTsiActiveSrcOsc0erClk OSCERCLK clock.

53.2.42.3.11 enum clock_tsi_lp_mode_src_k60d10_t

Enumerator

kClockTsiLpSrcLpoClk LPO clock.
kClockTsiLpSrcEr32kClk ERCLK32K clock.

53.2.42.3.12 enum clock_pllfl_sel_k60d10_t

Enumerator

kClockPllFllSelFll Fll clock.
kClockPllFllSelPll Pll0 clock.

53.2.42.3.13 enum clock_er32k_src_k60d10_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcRtc RTC 32k clock .
kClockEr32kSrcLpo LPO clock.

53.2.42.3.14 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.

53.2.42.3.15 enum clock_rtcout_src_k60d10_t

Enumerator

kClockRtcoutSrc1Hz 1Hz clock
kClockRtcoutSrc32kHz 32kHz clock

53.2.42.3.16 enum sim_adc_pretrg_sel_k60d10_t

Enumerator

kSimAdcPretrgselA Pre-trigger A selected for ADCx.
kSimAdcPretrgselB Pre-trigger B selected for ADCx.

53.2.42.3.17 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.

SIM HAL driver

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.

53.2.42.3.18 enum sim_uart_rxsrc_k60d10_t

Enumerator

kSimUartRxsrcPin UARTx_RX Pin.
kSimUartRxsrcCmp0 CMP0.
kSimUartRxsrcCmp1 CMP1.

53.2.42.3.19 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.

53.2.42.3.20 enum sim_ftm_trg_src_k60d10_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.
kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

53.2.42.3.21 enum sim_ftm_clk_sel_k60d10_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.
kSimFtmClkSel1 FTM CLKIN1 pin.

53.2.42.3.22 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.

53.2.42.3.23 enum sim_ftmflt_sel_k60d10_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

53.2.42.3.24 enum sim_tpm_clk_sel_k60d10_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM_CLKIN1 pin.

53.2.42.3.25 enum sim_tpm_ch_src_k60d10_t

Enumerator

- kSimTpmChSrc0* TPMx_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

53.2.42.3.26 enum sim_cmtuartpad_streng_h_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.

53.2.42.3.27 enum sim_ptd7pad_streng_h_k60d10_t

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

SIM HAL driver

53.2.42.3.28 enum sim_flexbus_security_level_k60d10_t

Enumerator

kSimFbslLevel0 FlexBus security level 0.

kSimFbslLevel1 FlexBus security level 1.

kSimFbslLevel2 FlexBus security level 2.

kSimFbslLevel3 FlexBus security level 3.

53.2.42.3.29 enum sim_clock_gate_name_k60d10_t

53.2.43 KV30F12810 SIM HAL driver

53.2.43.1 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](#) {
 [kClockPllFlSelIFll](#) = 0U,
 [kClockPllFlSelIrc48M](#) = 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 UART receive data source select.
- enum `sim_uart_txsrc_kv30f12810_t` {
 - kSimUartTxsrcPin,
 - kSimUartTxsrcFtm1,
 - kSimUartTxsrcFtm2 }

SIM UART 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_ftmflt_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.

53.2.43.2 Macro Definition Documentation

53.2.43.2.1 `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`

53.2.43.3 Enumeration Type Documentation

53.2.43.3.1 enum `clock_wdog_src_kv30f12810_t`

Enumerator

kClockWdogSrcLpoClk LPO.

kClockWdogSrcAltClk Alternative clock, for this SOC it is Bus clock.

53.2.43.3.2 enum `clock_trace_src_kv30f12810_t`

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.

kClockTraceSrcCoreClk core clock

53.2.43.3.3 enum `clock_port_filter_src_kv30f12810_t`

Enumerator

kClockPortFilterSrcBusClk Bus clock.

kClockPortFilterSrcLpoClk LPO.

SIM HAL driver

53.2.43.3.4 enum clock_lptmr_src_kv30f12810_t

Enumerator

kClockLptmrSrcMcgIrClk MCGIRCLK.
kClockLptmrSrcLpoClk LPO clock.
kClockLptmrSrcEr32kClk ERCLK32K clock.
kClockLptmrSrcOsc0erClkUndiv OSCERCLK_UNDIV clock.

53.2.43.3.5 enum clock_pllfl_sel_kv30f12810_t

Enumerator

kClockPlIFllSelFll Fll clock.
kClockPlIFllSelIrc48M IRC48MCLK.

53.2.43.3.6 enum clock_er32k_src_kv30f12810_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcLpo LPO clock.

53.2.43.3.7 enum clock_clkout_src_kv30f12810_t

Enumerator

kClockClkoutSelFlashClk Flash clock.
kClockClkoutSelLpoClk LPO clock.
kClockClkoutSelMcgIrClk MCGIRCLK.
kClockClkoutSelOsc0erClk OSC0ERCLK.
kClockClkoutSelIrc48M IRC48MCLK.

53.2.43.3.8 enum clock_osc32kout_sel_kv30f12810_t

Enumerator

kClockOsc32koutNone ERCLK32K is not output.
kClockOsc32koutPte0 ERCLK32K is output on PTE0.
kClockOsc32koutPte26 ERCLK32K is output on PTE26.

53.2.43.3.9 enum sim_adc_pretrg_sel_kv30f12810_t

Enumerator

kSimAdcPretrgSelA Pre-trigger A selected for ADCx.

kSimAdcPretrgSelB Pre-trigger B selected for ADCx.

53.2.43.3.10 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.

53.2.43.3.11 enum sim_uart_rxsrc_kv30f12810_t

Enumerator

kSimUartRxsrcPin UARTx_RX Pin.

kSimUartRxsrcCmp0 CMP0.

kSimUartRxsrcCmp1 CMP1.

53.2.43.3.12 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.

53.2.43.3.13 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

53.2.43.3.14 enum sim_ftm_clk_sel_kv30f12810_t

Enumerator

- kSimFtmClkSel0* FTM CLKIN0 pin.
- kSimFtmClkSel1* FTM CLKIN1 pin.

53.2.43.3.15 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.

53.2.43.3.16 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.

53.2.43.3.17 enum sim_ftmflt_sel_kv30f12810_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

53.2.43.3.18 enum sim_clock_gate_name_kv30f12810_t

53.2.44 KV31F12810 SIM HAL driver

53.2.44.1 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](#) {
[kClockPllFlSelFll](#) = 0U,
[kClockPllFlSelIrc48M](#) = 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_rxsrc_kv31f12810_t` {
`kSimLpuartRxsrcPin,`
`kSimLpuartRxsrcCmp0,`
`kSimLpuartRxsrcCmp1 }`
SIM LPUART RX source.
- enum `sim_uart_rxsrc_kv31f12810_t` {
`kSimUartRxsrcPin,`
`kSimUartRxsrcCmp0,`
`kSimUartRxsrcCmp1 }`
SIM UART receive data source select.
- enum `sim_uart_txsrc_kv31f12810_t` {
`kSimUartTxsrcPin,`
`kSimUartTxsrcFtm1,`
`kSimUartTxsrcFtm2 }`
SIM UART 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_ftmflt_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.

53.2.44.2 Macro Definition Documentation

53.2.44.2.1 `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`

53.2.44.3 Enumeration Type Documentation

53.2.44.3.1 enum `clock_wdog_src_kv31f12810_t`

Enumerator

kClockWdogSrcLpoClk LPO.

kClockWdogSrcAltClk Alternative clock, for this SOC it is Bus clock.

53.2.44.3.2 enum `clock_trace_src_kv31f12810_t`

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.

kClockTraceSrcCoreClk core clock

SIM HAL driver

53.2.44.3.3 enum clock_port_filter_src_kv31f12810_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.
kClockPortFilterSrcLpoClk LPO.

53.2.44.3.4 enum clock_lptmr_src_kv31f12810_t

Enumerator

kClockLptmrSrcMcgIrClk MCGIRCLK.
kClockLptmrSrcLpoClk LPO clock.
kClockLptmrSrcEr32kClk ERCLK32K clock.
kClockLptmrSrcOsc0erClkUndiv OSCERCLK_UNDIV clock.

53.2.44.3.5 enum clock_pllfl_sel_kv31f12810_t

Enumerator

kClockPlIFllSelFll Fll clock.
kClockPlIFllSelIrc48M IRC48MCLK.

53.2.44.3.6 enum clock_er32k_src_kv31f12810_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcLpo LPO clock.

53.2.44.3.7 enum clock_clkout_src_kv31f12810_t

Enumerator

kClockClkoutSelFlashClk Flash clock.
kClockClkoutSelLpoClk LPO clock.
kClockClkoutSelMcgIrClk MCGIRCLK.
kClockClkoutSelOsc0erClk OSC0ERCLK.
kClockClkoutSelIrc48M IRC48MCLK.

53.2.44.3.8 enum clock_osc32kout_sel_kv31f12810_t

Enumerator

- kClockOsc32koutNone* ERCLK32K is not output.
- kClockOsc32koutPte0* ERCLK32K is output on PTE0.
- kClockOsc32koutPte26* ERCLK32K is output on PTE26.

53.2.44.3.9 enum sim_adc_pretrg_sel_kv31f12810_t

Enumerator

- kSimAdcPretrgselA* Pre-trigger A selected for ADCx.
- kSimAdcPretrgselB* Pre-trigger B selected for ADCx.

53.2.44.3.10 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.

53.2.44.3.11 enum sim_lpuart_rxsrc_kv31f12810_t

Enumerator

- kSimLpuartRxsrcPin* LPUARTx_RX Pin.
- kSimLpuartRxsrcCmp0* CMP0.
- kSimLpuartRxsrcCmp1* CMP1.

53.2.44.3.12 enum sim_uart_rxsrc_kv31f12810_t

Enumerator

- kSimUartRxsrcPin* UARTx_RX Pin.

SIM HAL driver

kSimUartRxsSrcCmp0 CMP0.

kSimUartRxsSrcCmp1 CMP1.

53.2.44.3.13 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.

53.2.44.3.14 enum sim_ftm_trg_src_kv31f12810_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.

kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

53.2.44.3.15 enum sim_ftm_clk_sel_kv31f12810_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.

kSimFtmClkSel1 FTM CLKIN1 pin.

53.2.44.3.16 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.

53.2.44.3.17 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.

53.2.44.3.18 enum sim_ftm_ft_sel_kv31f12810_t

Enumerator

kSimFtmFtSel0 FlexTimer x fault y select 0.*kSimFtmFtSel1* FlexTimer x fault y select 1.**53.2.44.3.19 enum sim_clock_gate_name_kv31f12810_t**

SIM HAL driver

53.2.45 KV31F25612 SIM HAL driver

53.2.45.1 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 UART receive data source select.*

 - enum `sim_uart_txsrv_kv31f25612_t` {
`kSimUartTxsrvPin,`
`kSimUartTxsrvFtm1,`

SIM HAL driver

- `kSimUartTxsrcFtm2` }
SIM UART 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_ftmflt_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.

53.2.45.2 Macro Definition Documentation

53.2.45.2.1 `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`

53.2.45.3 Enumeration Type Documentation

53.2.45.3.1 enum `clock_wdog_src_kv31f25612_t`

Enumerator

kClockWdogSrcLpoClk LPO.

kClockWdogSrcAltClk Alternative clock, for this SOC it is Bus clock.

53.2.45.3.2 enum `clock_trace_src_kv31f25612_t`

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.

kClockTraceSrcCoreClk core clock

53.2.45.3.3 enum clock_port_filter_src_kv31f25612_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.
kClockPortFilterSrcLpoClk LPO.

53.2.45.3.4 enum clock_lptmr_src_kv31f25612_t

Enumerator

kClockLptmrSrcMcgIrClk MCGIRCLK.
kClockLptmrSrcLpoClk LPO clock.
kClockLptmrSrcEr32kClk ERCLK32K clock.
kClockLptmrSrcOsc0erClkUndiv OSCERCLK_UNDIV clock.

53.2.45.3.5 enum clock_lpuart_src_kv31f25612_t

Enumerator

kClockLpuartSrcNone Clock disabled.
kClockLpuartSrcPllFllSel Clock as selected by SOPT2[PLLFLLSEL].
kClockLpuartSrcOsc0erClk OSCERCLK.
kClockLpuartSrcMcgIrClk MCGIRCLK.

53.2.45.3.6 enum clock_pllfl_sel_kv31f25612_t

Enumerator

kClockPllFllSelFll Fll clock.
kClockPllFllSelPll Pll0 clock.
kClockPllFllSelIrc48M IRC48MCLK.

53.2.45.3.7 enum clock_er32k_src_kv31f25612_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcLpo LPO clock.

SIM HAL driver

53.2.45.3.8 enum clock_clkout_src_kv31f25612_t

Enumerator

kClockClkoutSelFlashClk Flash clock.
kClockClkoutSelLpoClk LPO clock.
kClockClkoutSelMcgIrClk MCGIRCLK.
kClockClkoutSelOsc0erClk OSC0ERCLK.
kClockClkoutSelIrc48M IRC48MCLK.

53.2.45.3.9 enum clock_osc32kout_sel_kv31f25612_t

Enumerator

kClockOsc32koutNone ERCLK32K is not output.
kClockOsc32koutPte0 ERCLK32K is output on PTE0.
kClockOsc32koutPte26 ERCLK32K is output on PTE26.

53.2.45.3.10 enum sim_adc_pretrg_sel_kv31f25612_t

Enumerator

kSimAdcPretrgSelA Pre-trigger A selected for ADCx.
kSimAdcPretrgSelB Pre-trigger B selected for ADCx.

53.2.45.3.11 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.

53.2.45.3.12 enum sim_lpuart_rxsrc_kv31f25612_t

Enumerator

- kSimLpuartRxsrcPin* LPUARTx_RX Pin.
- kSimLpuartRxsrcCmp0* CMP0.
- kSimLpuartRxsrcCmp1* CMP1.

53.2.45.3.13 enum sim_uart_rxsrc_kv31f25612_t

Enumerator

- kSimUartRxsrcPin* UARTx_RX Pin.
- kSimUartRxsrcCmp0* CMP0.
- kSimUartRxsrcCmp1* CMP1.

53.2.45.3.14 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.

53.2.45.3.15 enum sim_ftm_trg_src_kv31f25612_t

Enumerator

- kSimFtmTrgSrc0* FlexTimer x trigger y select 0.
- kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

53.2.45.3.16 enum sim_ftm_clk_sel_kv31f25612_t

Enumerator

- kSimFtmClkSel0* FTM CLKIN0 pin.
- kSimFtmClkSel1* FTM CLKIN1 pin.

SIM HAL driver

53.2.45.3.17 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.

53.2.45.3.18 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.

53.2.45.3.19 enum sim_ftmflt_sel_kv31f25612_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

53.2.45.3.20 enum sim_clock_gate_name_kv31f25612_t

53.2.46 KV31F51212 SIM HAL driver

53.2.46.1 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 UART receive data source select.*
 - enum `sim_uart_txsrv_kv31f51212_t` {
`kSimUartTxsrvPin,`
`kSimUartTxsrvFtm1,`
`kSimUartTxsrvFtm2 }`

- SIM UART 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_ft_sel_kv31f51212_t` {
`kSimFtmFltSel0`,
`kSimFtmFltSel1` }
- SIM FlexTimer x Fault y select.*

 - enum `sim_flexbus_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` (SIM_Type *base, `sim_clock_gate_name_t` name)
Enable the clock for specific module.
- static void `SIM_HAL_DisableClock` (SIM_Type *base, `sim_clock_gate_name_t` name)
Disable the clock for specific module.
- static bool `SIM_HAL_GetGateCmd` (SIM_Type *base, `sim_clock_gate_name_t` name)
Get the the clock gate state for specific module.
- static void `CLOCK_HAL_SetLpuartSrc` (SIM_Type *base, uint32_t instance, `clock_lpuart_src_t` setting)
Set LPUART clock source.
- static `clock_lpuart_src_t` `CLOCK_HAL_GetLpuartSrc` (SIM_Type *base, uint32_t instance)
Get LPUART clock source.
- static void `CLOCK_HAL_SetTraceClkSrc` (SIM_Type *base, `clock_trace_src_t` setting)
Set debug trace clock selection.
- static `clock_trace_src_t` `CLOCK_HAL_GetTraceClkSrc` (SIM_Type *base)

SIM HAL driver

- Get debug trace clock selection.*
- static void [CLOCK_HAL_SetExternalRefClock32kSrc](#) (SIM_Type *base, [clock_er32k_src_t](#) setting)
- Set the clock selection of ERCLK32K.*
- static [clock_er32k_src_t](#) [CLOCK_HAL_GetExternalRefClock32kSrc](#) (SIM_Type *base)
- Get the clock selection of ERCLK32K.*
- static void [CLOCK_HAL_SetPlllSel](#) (SIM_Type *base, [clock_pllfl_sel_t](#) setting)
- Set PLL/FLL clock selection.*
- static [clock_pllfl_sel_t](#) [CLOCK_HAL_GetPlllSel](#) (SIM_Type *base)
- Get PLL/FLL clock selection.*
- static void [CLOCK_HAL_SetClkOutSel](#) (SIM_Type *base, [clock_clkout_src_t](#) setting)
- Set CLKOUTSEL selection.*
- static [clock_clkout_src_t](#) [CLOCK_HAL_GetClkOutSel](#) (SIM_Type *base)
- Get CLKOUTSEL selection.*
- static void [CLOCK_HAL_SetOsc32kOutSel](#) (SIM_Type *base, [clock_osc32kout_sel_t](#) setting)
- Set OSC32KOUT selection.*
- static [clock_osc32kout_sel_t](#) [CLOCK_HAL_GetOsc32kOutSel](#) (SIM_Type *base)
- Get OSC32KOUT selection.*
- static void [CLOCK_HAL_SetOutDiv1](#) (SIM_Type *base, [uint8_t](#) setting)
- Set OUTDIV1.*
- static [uint8_t](#) [CLOCK_HAL_GetOutDiv1](#) (SIM_Type *base)
- Get OUTDIV1.*
- static void [CLOCK_HAL_SetOutDiv2](#) (SIM_Type *base, [uint8_t](#) setting)
- Set OUTDIV2.*
- static [uint8_t](#) [CLOCK_HAL_GetOutDiv2](#) (SIM_Type *base)
- Get OUTDIV2.*
- static void [CLOCK_HAL_SetOutDiv3](#) (SIM_Type *base, [uint8_t](#) setting)
- Set OUTDIV3.*
- static [uint8_t](#) [CLOCK_HAL_GetOutDiv3](#) (SIM_Type *base)
- Get OUTDIV3.*
- static void [CLOCK_HAL_SetOutDiv4](#) (SIM_Type *base, [uint8_t](#) setting)
- Set OUTDIV4.*
- static [uint8_t](#) [CLOCK_HAL_GetOutDiv4](#) (SIM_Type *base)
- Get OUTDIV4.*
- void [CLOCK_HAL_SetOutDiv](#) (SIM_Type *base, [uint8_t](#) outdiv1, [uint8_t](#) outdiv2, [uint8_t](#) outdiv3, [uint8_t](#) outdiv4)
- Sets the clock out dividers setting.*
- void [CLOCK_HAL_GetOutDiv](#) (SIM_Type *base, [uint8_t](#) *outdiv1, [uint8_t](#) *outdiv2, [uint8_t](#) *outdiv3, [uint8_t](#) *outdiv4)
- Gets the clock out dividers setting.*
- static [uint32_t](#) [SIM_HAL_GetRamSize](#) (SIM_Type *base)
- Gets RAM size.*
- static void [SIM_HAL_SetFlexbusSecurityLevelMode](#) (SIM_Type *base, [sim_flexbus_security_level_t](#) setting)
- Sets the FlexBus security level setting.*
- static [sim_flexbus_security_level_t](#) [SIM_HAL_GetFlexbusSecurityLevelMode](#) (SIM_Type *base)
- Gets the FlexBus security level setting.*
- void [SIM_HAL_SetAdcAlternativeTriggerCmd](#) (SIM_Type *base, [uint32_t](#) instance, bool enable)
- Sets the ADCx alternate trigger enable setting.*
- bool [SIM_HAL_GetAdcAlternativeTriggerCmd](#) (SIM_Type *base, [uint32_t](#) instance)
- Gets the ADCx alternate trigger enable setting.*

- void [SIM_HAL_SetAdcPreTriggerMode](#) (SIM_Type *base, uint32_t instance, [sim_adc_pretrg_sel_t](#) select)
Sets the ADCx pre-trigger select setting.
- [sim_adc_pretrg_sel_t SIM_HAL_GetAdcPreTriggerMode](#) (SIM_Type *base, uint32_t instance)
Gets the ADCx pre-trigger select setting.
- void [SIM_HAL_SetAdcTriggerMode](#) (SIM_Type *base, uint32_t instance, [sim_adc_trg_sel_t](#) select)
Sets the ADCx trigger select setting.
- [sim_adc_trg_sel_t SIM_HAL_GetAdcTriggerMode](#) (SIM_Type *base, uint32_t instance)
Gets the ADCx trigger select setting.
- void [SIM_HAL_SetAdcTriggerModeOneStep](#) (SIM_Type *base, 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](#) (SIM_Type *base, uint32_t instance, [sim_uart_rxsrc_t](#) select)
Sets the UARTx receive data source select setting.
- [sim_uart_rxsrc_t SIM_HAL_GetUartRxSrcMode](#) (SIM_Type *base, uint32_t instance)
Gets the UARTx receive data source select setting.
- void [SIM_HAL_SetUartTxSrcMode](#) (SIM_Type *base, uint32_t instance, [sim_uart_txsrc_t](#) select)
Sets the UARTx transmit data source select setting.
- [sim_uart_txsrc_t SIM_HAL_GetUartTxSrcMode](#) (SIM_Type *base, uint32_t instance)
Gets the UARTx transmit data source select setting.
- static void [SIM_HAL_SetLpuartRxSrcMode](#) (SIM_Type *base, uint32_t instance, [sim_lpuart_rxsrc_t](#) select)
Sets the LPUARTx receive data source select setting.
- static [sim_lpuart_rxsrc_t SIM_HAL_GetLpuartRxSrcMode](#) (SIM_Type *base, uint32_t instance)
Gets the LPUARTx receive data source select setting.
- void [SIM_HAL_SetFtmTriggerSrcMode](#) (SIM_Type *base, 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](#) (SIM_Type *base, uint32_t instance, uint8_t trigger)
Gets the FlexTimer x hardware trigger y source select setting.
- void [SIM_HAL_SetFtmExternalClkPinMode](#) (SIM_Type *base, 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](#) (SIM_Type *base, uint32_t instance)
Gets the FlexTimer x external clock pin select setting.
- void [SIM_HAL_SetFtmChSrcMode](#) (SIM_Type *base, 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](#) (SIM_Type *base, uint32_t instance, uint8_t channel)
Gets the FlexTimer x channel y input capture source select setting.
- void [SIM_HAL_SetFtmChOutSrcMode](#) (SIM_Type *base, 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](#) (SIM_Type *base, uint32_t instance, uint8_t channel)
Gets the FlexTimer x channel y output source select setting.
- void [SIM_HAL_SetFtmSyncCmd](#) (SIM_Type *base, uint32_t instance, bool sync)

SIM HAL driver

- Set FlexTimer x hardware trigger 0 software synchronization.*

 - static bool [SIM_HAL_GetFtmSyncCmd](#) (SIM_Type *base, uint32_t instance)
- Get FlexTimer x hardware trigger 0 software synchronization setting.*

 - void [SIM_HAL_SetFtmFaultSelMode](#) (SIM_Type *base, uint32_t instance, uint8_t fault, [sim_ftm_ft_sel_t](#) select)
- Sets the FlexTimer x fault y select setting.*

 - [sim_ftm_ft_sel_t SIM_HAL_GetFtmFaultSelMode](#) (SIM_Type *base, uint32_t instance, uint8_t fault)
- Gets the FlexTimer x fault y select setting.*

 - static uint32_t [SIM_HAL_GetFamilyId](#) (SIM_Type *base)
- Gets the Kinetis Family ID in the System Device ID register (SIM_SDID).*

 - static uint32_t [SIM_HAL_GetSubFamilyId](#) (SIM_Type *base)
- Gets the Kinetis Sub-Family ID in the System Device ID register (SIM_SDID).*

 - static uint32_t [SIM_HAL_GetSeriesId](#) (SIM_Type *base)
- Gets the Kinetis SeriesID in the System Device ID register (SIM_SDID).*

 - static uint32_t [SIM_HAL_GetPinCntId](#) (SIM_Type *base)
- Gets the Kinetis Pincount ID in System Device ID register (SIM_SDID).*

 - static uint32_t [SIM_HAL_GetRevId](#) (SIM_Type *base)
- Gets the Kinetis Revision ID in the System Device ID register (SIM_SDID).*

 - static uint32_t [SIM_HAL_GetDieId](#) (SIM_Type *base)
- Gets the Kinetis Die ID in the System Device ID register (SIM_SDID).*

 - static uint32_t [SIM_HAL_GetProgramFlashSize](#) (SIM_Type *base)
- Gets the program flash size in the Flash Configuration Register 1 (SIM_FCFG).*

 - static void [SIM_HAL_SetFlashDoze](#) (SIM_Type *base, uint32_t setting)
- Sets the Flash Doze in the Flash Configuration Register 1 (SIM_FCFG).*

 - static uint32_t [SIM_HAL_GetFlashDoze](#) (SIM_Type *base)
- Gets the Flash Doze in the Flash Configuration Register 1 (SIM_FCFG).*

 - static void [SIM_HAL_SetFlashDisableCmd](#) (SIM_Type *base, bool disable)
- Sets the Flash disable setting.*

 - static bool [SIM_HAL_GetFlashDisableCmd](#) (SIM_Type *base)
- Gets the Flash disable setting.*

 - static uint32_t [SIM_HAL_GetFlashMaxAddrBlock0](#) (SIM_Type *base)
- Gets the Flash maximum address block 0 in the Flash Configuration Register 1 (SIM_FCFG).*

 - static uint32_t [SIM_HAL_GetFlashMaxAddrBlock1](#) (SIM_Type *base)
- Gets the Flash maximum address block 1 in Flash Configuration Register 2.*

53.2.46.2 Macro Definition Documentation

53.2.46.2.1 #define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)

53.2.46.3 Enumeration Type Documentation

53.2.46.3.1 enum clock_wdog_src_kv31f51212_t

Enumerator

kClockWdogSrcLpoClk LPO.

kClockWdogSrcAltClk Alternative clock, for this SOC it is Bus clock.

53.2.46.3.2 enum clock_trace_src_kv31f51212_t

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.
kClockTraceSrcCoreClk core clock

53.2.46.3.3 enum clock_port_filter_src_kv31f51212_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.
kClockPortFilterSrcLpoClk LPO.

53.2.46.3.4 enum clock_lptmr_src_kv31f51212_t

Enumerator

kClockLptmrSrcMcgIrClk MCGIRCLK.
kClockLptmrSrcLpoClk LPO clock.
kClockLptmrSrcEr32kClk ERCLK32K clock.
kClockLptmrSrcOsc0erClkUndiv OSCERCLK_UNDIV clock.

53.2.46.3.5 enum clock_lpuart_src_kv31f51212_t

Enumerator

kClockLpuartSrcNone Clock disabled.
kClockLpuartSrcPllFllSel Clock as selected by SOPT2[PLLFLSEL].
kClockLpuartSrcOsc0erClk OSCERCLK.
kClockLpuartSrcMcgIrClk MCGIRCLK.

53.2.46.3.6 enum clock_pllfl_sel_kv31f51212_t

Enumerator

kClockPllFllSelFll Fll clock.
kClockPllFllSelPll Pll0 clock.
kClockPllFllSelIrc48M IRC48MCLK.

SIM HAL driver

53.2.46.3.7 enum clock_er32k_src_kv31f51212_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcLpo LPO clock.

53.2.46.3.8 enum clock_clkout_src_kv31f51212_t

Enumerator

kClockClkoutSelFlexbusClk Flexbus clock.
kClockClkoutSelFlashClk Flash clock.
kClockClkoutSelLpoClk LPO clock.
kClockClkoutSelMcgIrClk MCGIRCLK.
kClockClkoutSelOsc0erClk OSC0ERCLK.
kClockClkoutSelIrc48M IRC48MCLK.

53.2.46.3.9 enum sim_adc_pretrg_sel_kv31f51212_t

Enumerator

kSimAdcPretrgSelA Pre-trigger A selected for ADCx.
kSimAdcPretrgSelB Pre-trigger B selected for ADCx.

53.2.46.3.10 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.

53.2.46.3.11 enum sim_lpuart_rxsrc_kv31f51212_t

Enumerator

kSimLpuartRxsrcPin LPUARTx_RX Pin.
kSimLpuartRxsrcCmp0 CMP0.
kSimLpuartRxsrcCmp1 CMP1.

53.2.46.3.12 enum sim_uart_rxsrc_kv31f51212_t

Enumerator

kSimUartRxsrcPin UARTx_RX Pin.
kSimUartRxsrcCmp0 CMP0.
kSimUartRxsrcCmp1 CMP1.

53.2.46.3.13 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.

53.2.46.3.14 enum sim_ftm_trg_src_kv31f51212_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.
kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

53.2.46.3.15 enum sim_ftm_clk_sel_kv31f51212_t

Enumerator

kSimFtmClkSel0 FTM CLKIN0 pin.
kSimFtmClkSel1 FTM CLKIN1 pin.

SIM HAL driver

53.2.46.3.16 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.

53.2.46.3.17 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.

53.2.46.3.18 enum sim_ftmflt_sel_kv31f51212_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

53.2.46.3.19 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.

53.2.46.3.20 enum sim_clock_gate_name_kv31f51212_t

53.2.46.4 Function Documentation

53.2.46.4.1 static void SIM_HAL_EnableClock (SIM_Type * *base*, sim_clock_gate_name_t *name*) [inline], [static]

This function enables the clock for specific module.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>name</i>	Name of the module to enable.

53.2.46.4.2 `static void SIM_HAL_DisableClock (SIM_Type * base, sim_clock_gate_name_t name) [inline], [static]`

This function disables the clock for specific module.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>name</i>	Name of the module to disable.

53.2.46.4.3 `static bool SIM_HAL_GetGateCmd (SIM_Type * base, sim_clock_gate_name_t name) [inline], [static]`

This function will get the clock gate state for specific module.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>name</i>	Name of the module to get.

Returns

state true - ungated(Enabled), false - gated (Disabled)

53.2.46.4.4 `static void CLOCK_HAL_SetLpuartSrc (SIM_Type * base, uint32_t instance, clock_lpuart_src_t setting) [inline], [static]`

This function sets lpuart clock source selection.

Parameters

SIM HAL driver

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	LPUART instance.
<i>setting</i>	The value to set.

53.2.46.4.5 `static clock_lpuart_src_t CLOCK_HAL_GetLpuartSrc (SIM_Type * base, uint32_t instance) [inline], [static]`

This function gets lpuart clock source selection.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	LPUART instance.

Returns

Current selection.

53.2.46.4.6 `static void CLOCK_HAL_SetTraceClkSrc (SIM_Type * base, clock_trace_src_t setting) [inline], [static]`

This function sets debug trace clock selection.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

53.2.46.4.7 `static clock_trace_src_t CLOCK_HAL_GetTraceClkSrc (SIM_Type * base) [inline], [static]`

This function gets debug trace clock selection.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

Current selection.

53.2.46.4.8 `static void CLOCK_HAL_SetExternalRefClock32kSrc (SIM_Type * base,
clock_er32k_src_t setting) [inline], [static]`

This function sets the clock selection of ERCLK32K.

SIM HAL driver

Parameters

<i>base</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

53.2.46.4.9 `static clock_er32k_src_t CLOCK_HAL_GetExternalRefClock32kSrc (SIM_Type * base) [inline], [static]`

This function gets the clock selection of ERCLK32K.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

Current selection.

53.2.46.4.10 `static void CLOCK_HAL_SetPllflSel (SIM_Type * base, clock_pllfl_sel_t setting) [inline], [static]`

This function sets the selection of the high frequency clock for various peripheral clocking options

Parameters

<i>base</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

53.2.46.4.11 `static clock_pllfl_sel_t CLOCK_HAL_GetPllflSel (SIM_Type * base) [inline], [static]`

This function gets the selection of the high frequency clock for various peripheral clocking options

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

Current selection.


```
53.2.46.4.12 static void CLOCK_HAL_SetClkOutSel ( SIM_Type * base, 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>base</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

53.2.46.4.13 `static clock_clkout_src_t CLOCK_HAL_GetClkOutSel (SIM_Type * base)
[inline], [static]`

This function gets the selection of the clock to output on the CLKOUT pin.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

Current selection.

53.2.46.4.14 `static void CLOCK_HAL_SetOsc32kOutSel (SIM_Type * base,
clock_osc32kout_sel_t setting) [inline], [static]`

This function sets ERCLK32K output pin.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

53.2.46.4.15 `static clock_osc32kout_sel_t CLOCK_HAL_GetOsc32kOutSel (SIM_Type * base)
[inline], [static]`

This function gets ERCLK32K output pin setting.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

Current selection.

```
53.2.46.4.16 static void CLOCK_HAL_SetOutDiv1 ( SIM_Type * base, uint8_t setting )  
              [inline], [static]
```

This function sets divide value OUTDIV1.

SIM HAL driver

Parameters

<i>base</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

53.2.46.4.17 `static uint8_t CLOCK_HAL_GetOutDiv1 (SIM_Type * base) [inline], [static]`

This function gets divide value OUTDIV1.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

Current divide value.

53.2.46.4.18 `static void CLOCK_HAL_SetOutDiv2 (SIM_Type * base, uint8_t setting) [inline], [static]`

This function sets divide value OUTDIV2.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

53.2.46.4.19 `static uint8_t CLOCK_HAL_GetOutDiv2 (SIM_Type * base) [inline], [static]`

This function gets divide value OUTDIV2.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

Current divide value.

53.2.46.4.20 `static void CLOCK_HAL_SetOutDiv3 (SIM_Type * base, uint8_t setting)`
`[inline], [static]`

This function sets divide value OUTDIV3.

SIM HAL driver

Parameters

<i>base</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

53.2.46.4.21 `static uint8_t CLOCK_HAL_GetOutDiv3 (SIM_Type * base) [inline], [static]`

This function gets divide value OUTDIV3.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

Current divide value.

53.2.46.4.22 `static void CLOCK_HAL_SetOutDiv4 (SIM_Type * base, uint8_t setting) [inline], [static]`

This function sets divide value OUTDIV4.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>setting</i>	The value to set.

53.2.46.4.23 `static uint8_t CLOCK_HAL_GetOutDiv4 (SIM_Type * base) [inline], [static]`

This function gets divide value OUTDIV4.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

Current divide value.

53.2.46.4.24 void CLOCK_HAL_SetOutDiv (SIM_Type * *base*, 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>base</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

53.2.46.4.25 void **CLOCK_HAL_GetOutDiv** (SIM_Type * *base*, 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>base</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

53.2.46.4.26 static uint32_t **SIM_HAL_GetRamSize** (SIM_Type * *base*) [inline], [static]

This function gets the RAM size. The field specifies the amount of system RAM available on the device.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

size RAM size on the device

53.2.46.4.27 static void **SIM_HAL_SetFlexbusSecurityLevelMode** (SIM_Type * *base*, 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>base</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.

53.2.46.4.28 `static sim_flexbus_security_level_t SIM_HAL_GetFlexbusSecurityLevelMode (SIM_Type * base) [inline], [static]`

This function gets the FlexBus security level setting.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

setting FlexBus security level setting

53.2.46.4.29 `void SIM_HAL_SetAdcAlternativeTriggerCmd (SIM_Type * base, uint32_t instance, bool enable) [inline]`

This function enables/disables the alternative conversion triggers for ADCx.

Parameters

<i>base</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.

Sets the USB voltage regulator enabled setting.

This function enables/disables the alternative conversion triggers for ADCx.

SIM HAL driver

Parameters

<i>base</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.

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>base</i>	Base address for current SIM instance.
<i>enable</i>	USB voltage regulator enable setting <ul style="list-style-type: none"> • true: USB voltage regulator is enabled. • false: USB voltage regulator is disabled.

Sets the ADCx alternate trigger enable setting.

This function enables/disables the alternative conversion triggers for ADCx.

Parameters

<i>base</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.

53.2.46.4.30 `bool SIM_HAL_GetAdcAlternativeTriggerCmd (SIM_Type * base, uint32_t instance) [inline]`

This function gets the ADCx alternate trigger enable setting.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

Returns

enabled True if ADCx alternate trigger is enabled

53.2.46.4.31 void SIM_HAL_SetAdcPreTriggerMode (SIM_Type * *base*, 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>base</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>select</i>	pre-trigger select setting for ADCx

53.2.46.4.32 sim_adc_pretrg_sel_t SIM_HAL_GetAdcPreTriggerMode (SIM_Type * *base*, uint32_t *instance*) [inline]

This function gets the ADCx pre-trigger select setting.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

Returns

select ADCx pre-trigger select setting

53.2.46.4.33 void SIM_HAL_SetAdcTriggerMode (SIM_Type * *base*, uint32_t *instance*, sim_adc_trg_sel_t *select*) [inline]

This function selects the ADCx trigger source when alternative triggers are enabled through ADCxALTTRGEN.

SIM HAL driver

Parameters

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>select</i>	trigger select setting for ADCx

53.2.46.4.34 `sim_adc_trg_sel_t SIM_HAL_GetAdcTriggerMode (SIM_Type * base, uint32_t instance) [inline]`

This function gets the ADCx trigger select setting.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

Returns

ADCx trigger select setting

53.2.46.4.35 `void SIM_HAL_SetAdcTriggerModeOneStep (SIM_Type * base, 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>base</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.

53.2.46.4.36 `void SIM_HAL_SetUartRxSrcMode (SIM_Type * base, uint32_t instance, sim_uart_rxsrc_t select) [inline]`

This function selects the source for the UARTx receive data.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>select</i>	the source for the UARTx receive data

53.2.46.4.37 `sim_uart_rxsrc_t SIM_HAL_GetUartRxSrcMode (SIM_Type * base, uint32_t instance) [inline]`

This function gets the UARTx receive data source select setting.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

Returns

select UARTx receive data source select setting

53.2.46.4.38 `void SIM_HAL_SetUartTxSrcMode (SIM_Type * base, uint32_t instance, sim_uart_txsrc_t select) [inline]`

This function selects the source for the UARTx transmit data.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>select</i>	the source for the UARTx transmit data

53.2.46.4.39 `sim_uart_txsrc_t SIM_HAL_GetUartTxSrcMode (SIM_Type * base, uint32_t instance) [inline]`

This function gets the UARTx transmit data source select setting.

SIM HAL driver

Parameters

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

Returns

select UARTx transmit data source select setting

53.2.46.4.40 `static void SIM_HAL_SetLpuartRxSrcMode (SIM_Type * base, uint32_t instance, sim_lpuart_rxsrc_t select) [inline], [static]`

This function selects the source for the LPUARTx receive data.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>select</i>	the source for the LPUARTx receive data

53.2.46.4.41 `static sim_lpuart_rxsrc_t SIM_HAL_GetLpuartRxSrcMode (SIM_Type * base, uint32_t instance) [inline], [static]`

This function gets the LPUARTx receive data source select setting.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

Returns

select LPUARTx receive data source select setting

53.2.46.4.42 `void SIM_HAL_SetFtmTriggerSrcMode (SIM_Type * base, uint32_t instance, uint8_t trigger, sim_ftm_trg_src_t select)`

This function selects the source of FTMx hardware trigger y.

Parameters

<i>base</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.

53.2.46.4.43 `sim_ftm_trg_src_t SIM_HAL_GetFtmTriggerSrcMode (SIM_Type * base, uint32_t instance, uint8_t trigger)`

This function gets the FlexTimer x hardware trigger y source select setting.

Parameters

<i>base</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

53.2.46.4.44 `void SIM_HAL_SetFtmExternalClkPinMode (SIM_Type * base, uint32_t instance, sim_ftm_clk_sel_t select)`

This function selects the source of FTMx external clock pin select.

Parameters

<i>base</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.

SIM HAL driver

53.2.46.4.45 `sim_ftm_clk_sel_t SIM_HAL_GetFtmExternalClkPinMode (SIM_Type * base,
uint32_t instance)`

This function gets the FlexTimer x external clock pin select setting.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

Returns

select FlexTimer x external clock pin select setting

53.2.46.4.46 void SIM_HAL_SetFtmChSrcMode (SIM_Type * *base*, 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>base</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

53.2.46.4.47 sim_ftm_ch_src_t SIM_HAL_GetFtmChSrcMode (SIM_Type * *base*, uint32_t *instance*, uint8_t *channel*)

This function gets the FlexTimer x channel y input capture source select setting.

Parameters

<i>base</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

53.2.46.4.48 void SIM_HAL_SetFtmChOutSrcMode (SIM_Type * *base*, uint32_t *instance*, uint8_t *channel*, sim_ftm_ch_out_src_t *select*)

This function selects the FlexTimer x channel y output source.

SIM HAL driver

Parameters

<i>base</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

53.2.46.4.49 `sim_ftm_ch_out_src_t SIM_HAL_GetFtmChOutSrcMode (SIM_Type * base, uint32_t instance, uint8_t channel)`

This function gets the FlexTimer x channel y output source select setting.

Parameters

<i>base</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

53.2.46.4.50 `void SIM_HAL_SetFtmSyncCmd (SIM_Type * base, uint32_t instance, bool sync)`

This function sets FlexTimer x hardware trigger 0 software synchronization. FTMxSYNCBIT.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	device instance.
<i>sync</i>	Synchronize or not.

53.2.46.4.51 `static bool SIM_HAL_GetFtmSyncCmd (SIM_Type * base, uint32_t instance)
[inline], [static]`

This function gets FlexTimer x hardware trigger 0 software synchronization. FTMxSYNCBIT.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>instance</i>	device instance.

53.2.46.4.52 void **SIM_HAL_SetFtmFaultSelMode** (SIM_Type * *base*, uint32_t *instance*, uint8_t *fault*, sim_ftm_ft_sel_t *select*)

This function sets the FlexTimer x fault y select setting.

Parameters

<i>base</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.

53.2.46.4.53 sim_ftm_ft_sel_t **SIM_HAL_GetFtmFaultSelMode** (SIM_Type * *base*, uint32_t *instance*, uint8_t *fault*)

This function gets the FlexTimer x fault y select setting.

Parameters

<i>base</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

53.2.46.4.54 static uint32_t **SIM_HAL_GetFamilyId** (SIM_Type * *base*) [inline], [static]

This function gets the Kinetis Family ID in the System Device ID register.

SIM HAL driver

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

id Kinetis Family ID

53.2.46.4.55 `static uint32_t SIM_HAL_GetSubFamilyId (SIM_Type * base) [inline], [static]`

This function gets the Kinetis Sub-Family ID in System Device ID register.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

id Kinetis Sub-Family ID

53.2.46.4.56 `static uint32_t SIM_HAL_GetSeriesId (SIM_Type * base) [inline], [static]`

This function gets the Kinetis Series ID in System Device ID register.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

id Kinetis Series ID

53.2.46.4.57 `static uint32_t SIM_HAL_GetPinCntId (SIM_Type * base) [inline], [static]`

This function gets the Kinetis Pincount ID in System Device ID register.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

id Kinetis Pincount ID

53.2.46.4.58 `static uint32_t SIM_HAL_GetRevId (SIM_Type * base) [inline], [static]`

This function gets the Kinetis Revision ID in System Device ID register.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

id Kinetis Revision ID

53.2.46.4.59 `static uint32_t SIM_HAL_GetDieId (SIM_Type * base) [inline], [static]`

This function gets the Kinetis Die ID in System Device ID register.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

id Kinetis Die ID

53.2.46.4.60 `static uint32_t SIM_HAL_GetProgramFlashSize (SIM_Type * base) [inline], [static]`

This function gets the program flash size in the Flash Configuration Register 1.

SIM HAL driver

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

size Program flash Size

53.2.46.4.61 `static void SIM_HAL_SetFlashDoze (SIM_Type * base, uint32_t setting)`
[inline], [static]

This function sets the Flash Doze in the Flash Configuration Register 1.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>setting</i>	Flash Doze setting

53.2.46.4.62 `static uint32_t SIM_HAL_GetFlashDoze (SIM_Type * base)` **[inline],**
[static]

This function gets the Flash Doze in the Flash Configuration Register 1.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

setting Flash Doze setting

53.2.46.4.63 `static void SIM_HAL_SetFlashDisableCmd (SIM_Type * base, bool disable)`
[inline], [static]

This function sets the Flash disable setting in the Flash Configuration Register 1.

Parameters

<i>base</i>	Base address for current SIM instance.
<i>disable</i>	Flash disable setting

```
53.2.46.4.64 static bool SIM_HAL_GetFlashDisableCmd ( SIM_Type * base ) [inline],  
[static]
```

This function gets the Flash disable setting in the Flash Configuration Register 1.

SIM HAL driver

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

setting Flash disable setting

53.2.46.4.65 `static uint32_t SIM_HAL_GetFlashMaxAddrBlock0 (SIM_Type * base) [inline], [static]`

This function gets the Flash maximum block 0 in Flash Configuration Register 2.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

address Flash maximum block 0 address

53.2.46.4.66 `static uint32_t SIM_HAL_GetFlashMaxAddrBlock1 (SIM_Type * base) [inline], [static]`

This function gets the Flash maximum block 1 in Flash Configuration Register 1.

Parameters

<i>base</i>	Base address for current SIM instance.
-------------	--

Returns

address Flash maximum block 0 address

53.2.47 K63F12 SIM HAL driver

53.2.47.1 Overview

The section describes the enumerations, macros and data structures for K63F12 SIM HAL driver.

Files

- file [fsl_sim_hal_MK63F12.h](#)

Macros

- #define [FSL_SIM_SCGC_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)
SIM SCGC bit index.

Enumerations

- enum [clock_wdog_src_k63f12_t](#) {
[kClockWdogSrcLpoClk](#),
[kClockWdogSrcAltClk](#) }
WDOG clock source select.
- enum [clock_trace_src_k63f12_t](#) {
[kClockTraceSrcMcgoutClk](#),
[kClockTraceSrcCoreClk](#) }
Debug trace clock source select.
- enum [clock_port_filter_src_k63f12_t](#) {
[kClockPortFilterSrcBusClk](#),
[kClockPortFilterSrcLpoClk](#) }
PORTx digital input filter clock source select.
- enum [clock_lptmr_src_k63f12_t](#) {
[kClockLptmrSrcMcgIrClk](#),
[kClockLptmrSrcLpoClk](#),
[kClockLptmrSrcEr32kClk](#),
[kClockLptmrSrcOsc0erClk](#) }
LPTMR clock source select.
- enum [clock_time_src_k63f12_t](#) {
[kClockTimeSrcCoreSysClk](#),
[kClockTimeSrcPIIFilSel](#),
[kClockTimeSrcOsc0erClk](#),
[kClockTimeSrcExt](#) }
SIM timestamp clock source.
- enum [clock_rmii_src_k63f12_t](#) {
[kClockRmiiSrcExtalClk](#),
[kClockRmiiSrcExt](#) }
SIM RMII clock source.

SIM HAL driver

- enum `clock_usbfs_src_k63f12_t` {
`kClockUsbfsSrcExt`,
`kClockUsbfsSrcPllFllSel` }
SIM USB FS clock source.
- enum `clock_flexcan_src_k63f12_t` {
`kClockFlexcanSrcOsc0erClk`,
`kClockFlexcanSrcBusClk` }
FLEXCAN clock source select.
- enum `clock_sdhc_src_k63f12_t` {
`kClockSdhcSrcCoreSysClk`,
`kClockSdhcSrcPllFllSel`,
`kClockSdhcSrcOsc0erClk`,
`kClockSdhcSrcExt` }
SDHC clock source.
- enum `clock_sai_src_k63f12_t` {
`kClockSaiSrcSysClk` = 0U,
`kClockSaiSrcOsc0erClk` = 1U,
`kClockSaiSrcPllClk` = 3U }
SAI clock source.
- enum `clock_pllfl_sel_k63f12_t` {
`kClockPllFllSelFll` = 0U,
`kClockPllFllSelPll` = 1U,
`kClockPllFllSelIrc48M` = 3U }
SIM PLLFLLSEL clock source select.
- enum `clock_er32k_src_k63f12_t` {
`kClockEr32kSrcOsc0` = 0U,
`kClockEr32kSrcRtc` = 2U,
`kClockEr32kSrcLpo` = 3U }
SIM external reference clock source select (OSC32KSEL).
- enum `clock_clkout_src_k63f12_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_k63f12_t` {
`kClockRtcoutSrc1Hz`,
`kClockRtcoutSrc32kHz` }
SIM RTCCLKOUTSEL clock source select.
- enum `sim_usbsstby_mode_k63f12_t` {
`kSimUsbsstbyNoRegulator`,
`kSimUsbsstbyWithRegulator` }
SIM USB voltage regulator in standby mode setting during stop modes.
- enum `sim_usbvstby_mode_k63f12_t` {

- kSimUsbvstbyNoRegulator,
 - kSimUsbvstbyWithRegulator }
- SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.*
 - enum sim_adc_pretrg_sel_k63f12_t {
 - kSimAdcPretrgselA,
 - kSimAdcPretrgselB }
 - SIM ADCx pre-trigger select.*
 - enum sim_adc_trg_sel_k63f12_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_k63f12_t {
 - kSimUartRxsrcPin,
 - kSimUartRxsrcCmp0,
 - kSimUartRxsrcCmp1 }
 - SIM UART receive data source select.*
 - enum sim_uart_txsrc_k63f12_t {
 - kSimUartTxsrcPin,
 - kSimUartTxsrcFtm1,
 - kSimUartTxsrcFtm2 }
 - SIM UART transmit data source select.*
 - enum sim_ftm_trg_src_k63f12_t {
 - kSimFtmTrgSrc0,
 - kSimFtmTrgSrc1 }
 - SIM FlexTimer x trigger y select.*
 - enum sim_ftm_clk_sel_k63f12_t {
 - kSimFtmClkSel0,
 - kSimFtmClkSel1 }
 - SIM FlexTimer external clock select.*
 - enum sim_ftm_ch_src_k63f12_t {
 - kSimFtmChSrc0,
 - kSimFtmChSrc1,
 - kSimFtmChSrc2,

SIM HAL driver

- `kSimFtmChSrc3` }
 - SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftm_ft_sel_k63f12_t` {
 - `kSimFtmFltSel0`,
 - `kSimFtmFltSel1` }
 - SIM FlexTimer x Fault y select.*
 - enum `sim_tpm_clk_sel_k63f12_t` {
 - `kSimTpmClkSel0`,
 - `kSimTpmClkSel1` }
 - SIM Timer/PWM external clock select.*
 - enum `sim_tpm_ch_src_k63f12_t` {
 - `kSimTpmChSrc0`,
 - `kSimTpmChSrc1`,
 - `kSimTpmChSrc2`,
 - `kSimTpmChSrc3` }
 - SIM Timer/PWM x channel y input capture source select.*
 - enum `sim_cmtuartpad_strength_k63f12_t` {
 - `kSimCmtuartSinglePad`,
 - `kSimCmtuartDualPad` }
 - SIM CMT/UART pad drive strength.*
 - enum `sim_ptd7pad_strength_k63f12_t` {
 - `kSimPtd7padSinglePad`,
 - `kSimPtd7padDualPad` }
 - SIM PTD7 pad drive strength.*
 - enum `sim_flexbus_security_level_k63f12_t` {
 - `kSimFbslLevel0`,
 - `kSimFbslLevel1`,
 - `kSimFbslLevel2`,
 - `kSimFbslLevel3` }
 - SIM FlexBus security level.*
 - enum `sim_clock_gate_name_k63f12_t`
 - Clock gate name used for SIM_HAL_EnableClock/SIM_HAL_DisableClock.*

53.2.47.2 Macro Definition Documentation

53.2.47.2.1 `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`

53.2.47.3 Enumeration Type Documentation

53.2.47.3.1 enum `clock_wdog_src_k63f12_t`

Enumerator

`kClockWdogSrcLpoClk` LPO.

`kClockWdogSrcAltClk` Alternative clock, for MK63F12 it is Bus clock.

53.2.47.3.2 enum clock_trace_src_k63f12_t

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.
kClockTraceSrcCoreClk core clock

53.2.47.3.3 enum clock_port_filter_src_k63f12_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.
kClockPortFilterSrcLpoClk LPO.

53.2.47.3.4 enum clock_lptmr_src_k63f12_t

Enumerator

kClockLptmrSrcMcgIrClk MCGIRCLK.
kClockLptmrSrcLpoClk LPO clock.
kClockLptmrSrcEr32kClk ERCLK32K clock.
kClockLptmrSrcOsc0erClk OSCERCLK clock.

53.2.47.3.5 enum clock_time_src_k63f12_t

Enumerator

kClockTimeSrcCoreSysClk Core/system clock.
kClockTimeSrcPlLFllSel clock as selected by SOPT2[PLLFLLSEL].
kClockTimeSrcOsc0erClk OSCERCLK clock.
kClockTimeSrcExt ENET 1588 clock in (ENET_1588_CLKIN)

53.2.47.3.6 enum clock_rmii_src_k63f12_t

Enumerator

kClockRmiiSrcExtalClk EXTAL Clock.
kClockRmiiSrcExt ENET 1588 clock in (ENET_1588_CLKIN)

53.2.47.3.7 enum clock_usbfs_src_k63f12_t

Enumerator

kClockUsbfsSrcExt External bypass clock (USB_CLKIN)
kClockUsbfsSrcPlLFllSel Clock divider USB FS clock.

SIM HAL driver

53.2.47.3.8 enum clock_flexcan_src_k63f12_t

Enumerator

kClockFlexcanSrcOsc0erClk OSCERCLK.
kClockFlexcanSrcBusClk Bus clock.

53.2.47.3.9 enum clock_sdhc_src_k63f12_t

Enumerator

kClockSdhcSrcCoreSysClk Core/system clock.
kClockSdhcSrcPllFllSel clock as selected by SOPT2[PLLFLSEL].
kClockSdhcSrcOsc0erClk OSCERCLK clock.
kClockSdhcSrcExt External bypass clock (SDHC0_CLKIN)

53.2.47.3.10 enum clock_sai_src_k63f12_t

Enumerator

kClockSaiSrcSysClk SYSCLK.
kClockSaiSrcOsc0erClk OSC0ERCLK.
kClockSaiSrcPllClk MCGPLLCLK.

53.2.47.3.11 enum clock_pllfl_sel_k63f12_t

Enumerator

kClockPllFlSelFll Fll clock.
kClockPllFlSelPll Pll0 clock.
kClockPllFlSelIrc48M IRC48MCLK.

53.2.47.3.12 enum clock_er32k_src_k63f12_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcRtc RTC 32k clock .
kClockEr32kSrcLpo LPO clock.

53.2.47.3.13 enum clock_clkout_src_k63f12_t

Enumerator

kClockClkoutSelFlexbusClk Flexbus clock.
kClockClkoutSelFlashClk Flash clock.
kClockClkoutSelLpoClk LPO clock.
kClockClkoutSelMcgIrClk MCGIRCLK.
kClockClkoutSelRtc RTC 32k clock.
kClockClkoutSelOsc0erClk OSC0ERCLK.
kClockClkoutSelIrc48M IRC48MCLK.

53.2.47.3.14 enum clock_rtcout_src_k63f12_t

Enumerator

kClockRtcoutSrc1Hz 1Hz clock
kClockRtcoutSrc32kHz 32kHz clock

53.2.47.3.15 enum sim_usbsstby_mode_k63f12_t

Enumerator

kSimUsbsstbyNoRegulator regulator not in standby during Stop modes
kSimUsbsstbyWithRegulator regulator in standby during Stop modes

53.2.47.3.16 enum sim_usbvstby_mode_k63f12_t

Enumerator

kSimUsbvstbyNoRegulator regulator not in standby during VLPR and VLPW modes
kSimUsbvstbyWithRegulator regulator in standby during VLPR and VLPW modes

53.2.47.3.17 enum sim_adc_pretrg_sel_k63f12_t

Enumerator

kSimAdcPretrgSelA Pre-trigger A selected for ADCx.
kSimAdcPretrgSelB Pre-trigger B selected for ADCx.

SIM HAL driver

53.2.47.3.18 enum sim_adc_trg_sel_k63f12_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.

53.2.47.3.19 enum sim_uart_rxsrc_k63f12_t

Enumerator

kSimUartRxsrcPin UARTx_RX Pin.
kSimUartRxsrcCmp0 CMP0.
kSimUartRxsrcCmp1 CMP1.

53.2.47.3.20 enum sim_uart_txsrc_k63f12_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.

53.2.47.3.21 enum sim_ftm_trg_src_k63f12_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.
kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

53.2.47.3.22 enum sim_ftm_clk_sel_k63f12_t

Enumerator

- kSimFtmClkSel0* FTM CLKIN0 pin.
- kSimFtmClkSel1* FTM CLKIN1 pin.

53.2.47.3.23 enum sim_ftm_ch_src_k63f12_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.

53.2.47.3.24 enum sim_ftmflt_sel_k63f12_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

53.2.47.3.25 enum sim_tpm_clk_sel_k63f12_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM_CLKIN1 pin.

53.2.47.3.26 enum sim_tpm_ch_src_k63f12_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.

53.2.47.3.27 enum sim_cmtuartpad_strength_k63f12_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

53.2.47.3.28 enum sim_ptd7pad_strength_k63f12_t

Enumerator

kSimPtd7padSinglePad Single-pad drive strength for PTD7.

kSimPtd7padDualPad Dual-pad drive strength for PTD7.

53.2.47.3.29 enum sim_flexbus_security_level_k63f12_t

Enumerator

kSimFbslLevel0 FlexBus security level 0.

kSimFbslLevel1 FlexBus security level 1.

kSimFbslLevel2 FlexBus security level 2.

kSimFbslLevel3 FlexBus security level 3.

53.2.47.3.30 enum sim_clock_gate_name_k63f12_t

53.2.48 K64F12 SIM HAL driver

53.2.48.1 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](#),
[kClockTimeSrcPIIFilSel](#),
[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` {
`kSimUsbsstbyNoRegulator`,
`kSimUsbsstbyWithRegulator` }
SIM USB voltage regulator in standby mode setting during stop modes.
- enum `sim_usbvstby_mode_k64f12_t` {

kSimUsbvstbyNoRegulator,
kSimUsbvstbyWithRegulator }

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 UART receive data source select.

- enum sim_uart_txsrc_k64f12_t {
kSimUartTxsrcPin,
kSimUartTxsrcFtm1,
kSimUartTxsrcFtm2 }

SIM UART 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_ft_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.

53.2.48.2 Macro Definition Documentation

53.2.48.2.1 `#define FSL_SIM_SCGC_BIT(SCGCx, n) (((SCGCx-1U)<<5U) + n)`

53.2.48.3 Enumeration Type Documentation

53.2.48.3.1 enum `clock_wdog_src_k64f12_t`

Enumerator

`kClockWdogSrcLpoClk` LPO.

`kClockWdogSrcAltClk` Alternative clock, for K64F12 it is Bus clock.

53.2.48.3.2 enum clock_trace_src_k64f12_t

Enumerator

kClockTraceSrcMcgoutClk MCG out clock.
kClockTraceSrcCoreClk core clock

53.2.48.3.3 enum clock_port_filter_src_k64f12_t

Enumerator

kClockPortFilterSrcBusClk Bus clock.
kClockPortFilterSrcLpoClk LPO.

53.2.48.3.4 enum clock_lptmr_src_k64f12_t

Enumerator

kClockLptmrSrcMcgIrClk MCGIRCLK.
kClockLptmrSrcLpoClk LPO clock.
kClockLptmrSrcEr32kClk ERCLK32K clock.
kClockLptmrSrcOsc0erClk OSCERCLK clock.

53.2.48.3.5 enum clock_time_src_k64f12_t

Enumerator

kClockTimeSrcCoreSysClk Core/system clock.
kClockTimeSrcPlLFllSel clock as selected by SOPT2[PLLFLLSEL].
kClockTimeSrcOsc0erClk OSCERCLK clock.
kClockTimeSrcExt ENET 1588 clock in (ENET_1588_CLKIN)

53.2.48.3.6 enum clock_rmii_src_k64f12_t

Enumerator

kClockRmiiSrcExtalClk EXTAL Clock.
kClockRmiiSrcExt ENET 1588 clock in (ENET_1588_CLKIN)

53.2.48.3.7 enum clock_usbfs_src_k64f12_t

Enumerator

kClockUsbfsSrcExt External bypass clock (USB_CLKIN)
kClockUsbfsSrcPlLFllSel Clock divider USB FS clock.

SIM HAL driver

53.2.48.3.8 enum clock_flexcan_src_k64f12_t

Enumerator

kClockFlexcanSrcOsc0erClk OSCERCLK.
kClockFlexcanSrcBusClk Bus clock.

53.2.48.3.9 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)

53.2.48.3.10 enum clock_sai_src_k64f12_t

Enumerator

kClockSaiSrcSysClk SYSCLK.
kClockSaiSrcOsc0erClk OSC0ERCLK.
kClockSaiSrcPllClk MCGPLLCLK.

53.2.48.3.11 enum clock_pllfl_sel_k64f12_t

Enumerator

kClockPllFlSelFll Fll clock.
kClockPllFlSelPll Pll0 clock.
kClockPllFlSelIrc48M IRC48MCLK.

53.2.48.3.12 enum clock_er32k_src_k64f12_t

Enumerator

kClockEr32kSrcOsc0 OSC0 clock (OSC032KCLK).
kClockEr32kSrcRtc RTC 32k clock .
kClockEr32kSrcLpo LPO clock.

53.2.48.3.13 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.

53.2.48.3.14 enum clock_rtcout_src_k64f12_t

Enumerator

kClockRtcoutSrc1Hz 1Hz clock
kClockRtcoutSrc32kHz 32kHz clock

53.2.48.3.15 enum sim_usbsstby_mode_k64f12_t

Enumerator

kSimUsbsstbyNoRegulator regulator not in standby during Stop modes
kSimUsbsstbyWithRegulator regulator in standby during Stop modes

53.2.48.3.16 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

53.2.48.3.17 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

53.2.48.3.18 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.

53.2.48.3.19 enum sim_uart_rxsrc_k64f12_t

Enumerator

kSimUartRxsrcPin UARTx_RX Pin.
kSimUartRxsrcCmp0 CMP0.
kSimUartRxsrcCmp1 CMP1.

53.2.48.3.20 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.

53.2.48.3.21 enum sim_ftm_trg_src_k64f12_t

Enumerator

kSimFtmTrgSrc0 FlexTimer x trigger y select 0.
kSimFtmTrgSrc1 FlexTimer x trigger y select 1.

53.2.48.3.22 enum sim_ftm_clk_sel_k64f12_t

Enumerator

- kSimFtmClkSel0* FTM CLKIN0 pin.
- kSimFtmClkSel1* FTM CLKIN1 pin.

53.2.48.3.23 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.

53.2.48.3.24 enum sim_ftmflt_sel_k64f12_t

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

53.2.48.3.25 enum sim_tpm_clk_sel_k64f12_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM_CLKIN1 pin.

53.2.48.3.26 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.

53.2.48.3.27 enum sim_cmtuartpad_strength_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

53.2.48.3.28 enum sim_ptd7pad_strength_k64f12_t

Enumerator

kSimPtd7padSinglePad Single-pad drive strength for PTD7.

kSimPtd7padDualPad Dual-pad drive strength for PTD7.

53.2.48.3.29 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.

53.2.48.3.30 enum sim_clock_gate_name_k64f12_t

53.2.49 K65F18 SIM HAL driver

53.2.49.1 Overview

The section describes the enumerations, macros and data structures for K65F18 SIM HAL driver.

Files

- file [fsl_sim_hal_MK65F18.h](#)

Macros

- #define [FSL_SIM_SCGC_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)
SIM SCGC bit index.

Enumerations

- enum [clock_wdog_src_k65f18_t](#) {
[kClockWdogSrcLpoClk](#),
[kClockWdogSrcAltClk](#) }
WDOG clock source select.
- enum [clock_trace_src_k65f18_t](#) {
[kClockTraceSrcMcgoutClkDiv](#),
[kClockTraceSrcCoreClk](#) }
Debug trace clock source select.
- enum [clock_port_filter_src_k65f18_t](#) {
[kClockPortFilterSrcBusClk](#),
[kClockPortFilterSrcLpoClk](#) }
PORTx digital input filter clock source select.
- enum [clock_tpm_src_k65f18_t](#) {
[kClockTpmSrcNone](#),
[kClockTpmSrcPlIFllSelDiv](#),
[kClockTpmSrcOsc0erClk](#),
[kClockTpmSrcMcgIrClk](#) }
TPM clock source select.
- enum [clock_lptmr_src_k65f18_t](#) {
[kClockLptmrSrcMcgIrClk](#),
[kClockLptmrSrcLpoClk](#),
[kClockLptmrSrcEr32kClk](#),
[kClockLptmrSrcOsc0erClkUndiv](#) }
LPTMR clock source select.
- enum [clock_lpuart_src_k65f18_t](#) {
[kClockLpuartSrcNone](#),
[kClockLpuartSrcPlIFllSel](#),
[kClockLpuartSrcOsc0erClk](#),

SIM HAL driver

- `kClockLpuartSrcMcgIrClk` }

SIM LPUART clock source.
- enum `sim_lpuart_rxsrc_k65f18_t` {
 - `kSimLpuartRxsSrcPin`,
 - `kSimLpuartRxsSrcCmp0`,
 - `kSimLpuartRxsSrcCmp1` }
 - SIM LPUART RX source.*
- enum `sim_lpuart_txsrc_k65f18_t` {
 - `kSimLpuartTxsrcPin`,
 - `kSimLpuartTxsrcTpm1`,
 - `kSimLpuartTxsrcTpm2` }
 - SIM LPUART TX source.*
- enum `clock_time_src_k65f18_t` {
 - `kClockTimeSrcCoreSysClk`,
 - `kClockTimeSrcPllFllSel`,
 - `kClockTimeSrcOsc0erClk`,
 - `kClockTimeSrcExt` }
 - SIM timestamp clock source.*
- enum `clock_rmii_src_k65f18_t` {
 - `kClockRmiiSrcExtalClk`,
 - `kClockRmiiSrcExt` }
 - SIM RMII clock source.*
- enum `clock_usbfs_src_k65f18_t` {
 - `kClockUsbfsSrcExt`,
 - `kClockUsbfsSrcPllFllSel` }
 - SIM USB FS clock source.*
- enum `clock_flexcan_src_k65f18_t` {
 - `kClockFlexcanSrcOsc0erClk`,
 - `kClockFlexcanSrcBusClk` }
 - FLEXCAN clock source select.*
- enum `clock_sdhc_src_k65f18_t` {
 - `kClockSdhcSrcCoreSysClk`,
 - `kClockSdhcSrcPllFllSel`,
 - `kClockSdhcSrcOsc0erClk`,
 - `kClockSdhcSrcExt` }
 - SDHC clock source.*
- enum `clock_sai_src_k65f18_t` {
 - `kClockSaiSrcSysClk` = 0U,
 - `kClockSaiSrcOsc0erClk` = 1U,
 - `kClockSaiSrcPllFllSel` = 3U }
 - SAI clock source.*
- enum `clock_pllfl_sel_k65f18_t` {
 - `kClockPllFllSelFll` = 0U,
 - `kClockPllFllSelPll` = 1U,
 - `kClockPllFllSelUsb1pfd` = 2U,
 - `kClockPllFllSelIrc48M` = 3U }
 - SIM PLLFLLSEL clock source select.*

- enum `clock_er32k_src_k65f18_t` {
`kClockEr32kSrcOsc0 = 0U,`
`kClockEr32kSrcRtc = 2U,`
`kClockEr32kSrcLpo = 3U }`
SIM external reference clock source select (OSC32KSEL).
- enum `clock_clkout_src_k65f18_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_k65f18_t` {
`kClockRtcoutSrc1Hz,`
`kClockRtcoutSrc32kHz }`
SIM RTCCLKOUTSEL clock source select.
- enum `clock_usbhs_slowclk_src_k65f18_t` {
`kClockUsbhsSlowClkSrcMcgIrClk,`
`kClockUsbhsSlowClkSrcRtc32kHz }`
SIM USBHS/USBPHY slow clock source select.
- enum `sim_usbsstby_mode_k65f18_t` {
`kSimUsbsstbyNoRegulator,`
`kSimUsbsstbyWithRegulator }`
SIM USB voltage regulator in standby mode setting during stop modes.
- enum `sim_usbvstby_mode_k65f18_t` {
`kSimUsbvstbyNoRegulator,`
`kSimUsbvstbyWithRegulator }`
SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.
- enum `sim_usbvout_mode_k65f18_t` {
`kSimUsbvout2_733V,`
`kSimUsbvout3_020V,`
`kSimUsbvout3_074V,`
`kSimUsbvout3_130V,`
`kSimUsbvout3_188V,`
`kSimUsbvout3_248V,`
`kSimUsbvout3_310V }`
SIM USB voltage regulator 3.3 output target.
- enum `sim_adc_pretrg_sel_k65f18_t` {
`kSimAdcPretrgselA,`
`kSimAdcPretrgselB }`
SIM ADCx pre-trigger select.
- enum `sim_adc_trg_sel_k65f18_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,
kSimAdcTrgSelTpm = 15U }

```
- SIM ADCx trigger select.*

```

enum sim_uart_rxsrc_k65f18_t {
kSimUartRxsrcPin,
kSimUartRxsrcCmp0,
kSimUartRxsrcCmp1 }

```

*SIM UART receive data source select.*

```

enum sim_uart_txsrc_k65f18_t {
kSimUartTxsrcPin,
kSimUartTxsrcFtm1,
kSimUartTxsrcFtm2 }

```

*SIM UART transmit data source select.*

```

enum sim_ftm_trg_src_k65f18_t {
kSimFtmTrgSrc0,
kSimFtmTrgSrc1 }

```

*SIM FlexTimer x trigger y select.*

```

enum sim_ftm_clk_sel_k65f18_t {
kSimFtmClkSel0,
kSimFtmClkSel1 }

```

*SIM FlexTimer external clock select.*

```

enum sim_ftm_ch_src_k65f18_t {
kSimFtmChSrc0,
kSimFtmChSrc1,
kSimFtmChSrc2,
kSimFtmChSrc3 }

```

*SIM FlexTimer x channel y input capture source select.*

```

enum sim_ftm_ch_out_src_k65f18_t {
kSimFtmChOutSrc0,
kSimFtmChOutSrc1 }

```

*SIM FlexTimer x channel y output source select.*

```

enum sim_ftm_ft_sel_k65f18_t {

```



- kSimFtmFltSel0,
  - kSimFtmFltSel1 }
    - SIM FlexTimer x Fault y select.*
  - enum `sim_tpm_clk_sel_k65f18_t` {
    - kSimTpmClkSel0,
    - kSimTpmClkSel1 }
      - SIM Timer/PWM external clock select.*
    - enum `sim_tpm_ch_src_k65f18_t` {
      - kSimTpmChSrc0,
      - kSimTpmChSrc1,
      - kSimTpmChSrc2,
      - kSimTpmChSrc3 }
        - SIM Timer/PWM x channel y input capture source select.*
      - enum `sim_cmtuartpad_strength_k65f18_t` {
        - kSimCmtuartSinglePad,
        - kSimCmtuartDualPad }
          - SIM CMT/UART pad drive strength.*
        - enum `sim_ptd7pad_strength_k65f18_t` {
          - kSimPtd7padSinglePad,
          - kSimPtd7padDualPad }
            - SIM PTD7 pad drive strength.*
          - enum `sim_flexbus_security_level_k65f18_t` {
            - kSimFbslLevel0,
            - kSimFbslLevel1,
            - kSimFbslLevel2,
            - kSimFbslLevel3 }
              - SIM FlexBus security level.*
            - enum `sim_clock_gate_name_k65f18_t`
              - Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

## 53.2.49.2 Macro Definition Documentation

53.2.49.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

## 53.2.49.3 Enumeration Type Documentation

53.2.49.3.1 `enum clock_wdog_src_k65f18_t`

Enumerator

*kClockWdogSrcLpoClk* LPO.

*kClockWdogSrcAltClk* Alternative clock, for this SOC it is Bus clock.

## SIM HAL driver

### 53.2.49.3.2 enum clock\_trace\_src\_k65f18\_t

Enumerator

*kClockTraceSrcMcgoutClkDiv* MCG out clock divided by the fractional divider configured by SIM\_CLKDIV4[TRACEFRAC, TRACEDIV].  
*kClockTraceSrcCoreClk* core clock

### 53.2.49.3.3 enum clock\_port\_filter\_src\_k65f18\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.  
*kClockPortFilterSrcLpoClk* LPO.

### 53.2.49.3.4 enum clock\_tpm\_src\_k65f18\_t

Enumerator

*kClockTpmSrcNone* clock disabled  
*kClockTpmSrcPllFllSelDiv* clock as selected by SOPT2[PLLFLSEL] and divided by the fractional divider configured by SIM\_CLKDIV3[PLLFLFRAC, PLLFLLDIV].  
*kClockTpmSrcOsc0erClk* OSCERCLK clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.49.3.5 enum clock\_lptmr\_src\_k65f18\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 53.2.49.3.6 enum clock\_lpuart\_src\_k65f18\_t

Enumerator

*kClockLpuartSrcNone* Clock disabled.  
*kClockLpuartSrcPllFllSel* Clock as selected by SOPT2[PLLFLSEL].  
*kClockLpuartSrcOsc0erClk* OSCERCLK.  
*kClockLpuartSrcMcgIrClk* MCGIRCLK.

**53.2.49.3.7 enum sim\_lpuart\_rxsrc\_k65f18\_t**

Enumerator

*kSimLpuartRxsrcPin* LPUARTx\_RX Pin.  
*kSimLpuartRxsrcCmp0* CMP0.  
*kSimLpuartRxsrcCmp1* CMP1.

**53.2.49.3.8 enum sim\_lpuart\_txsrc\_k65f18\_t**

Enumerator

*kSimLpuartTxsrcPin* UARTx\_TX Pin.  
*kSimLpuartTxsrcTpm1* UARTx\_TX pin modulated with TPM1 channel 0 output.  
*kSimLpuartTxsrcTpm2* UARTx\_TX pin modulated with TPM2 channel 0 output.

**53.2.49.3.9 enum clock\_time\_src\_k65f18\_t**

Enumerator

*kClockTimeSrcCoreSysClk* Core/system clock.  
*kClockTimeSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockTimeSrcOsc0erClk* OSCERCLK clock.  
*kClockTimeSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

**53.2.49.3.10 enum clock\_rmii\_src\_k65f18\_t**

Enumerator

*kClockRmiiSrcExtalClk* EXTAL Clock.  
*kClockRmiiSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

**53.2.49.3.11 enum clock\_usbfs\_src\_k65f18\_t**

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)  
*kClockUsbfsSrcPllFllSel* Clock divider USB FS clock.

**53.2.49.3.12 enum clock\_flexcan\_src\_k65f18\_t**

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.

## SIM HAL driver

### 53.2.49.3.13 enum clock\_sdhc\_src\_k65f18\_t

Enumerator

*kClockSdhcSrcCoreSysClk* Core/system clock.  
*kClockSdhcSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockSdhcSrcOsc0erClk* OSCERCLK clock.  
*kClockSdhcSrcExt* External bypass clock (SDHC0\_CLKIN)

### 53.2.49.3.14 enum clock\_sai\_src\_k65f18\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllFllSel* MCGPLLFLCLK.

### 53.2.49.3.15 enum clock\_pllfl\_sel\_k65f18\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.  
*kClockPllFllSelUsb1pfd* USB1 PFD clock.  
*kClockPllFllSelIrc48M* IRC48MCLK.

### 53.2.49.3.16 enum clock\_er32k\_src\_k65f18\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.49.3.17 enum clock\_clkout\_src\_k65f18\_t

Enumerator

*kClockClkoutSelFlexbusClk* Flexbus clock.  
*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCGIRCLK.  
*kClockClkoutSelRtc* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSC0ERCLK.  
*kClockClkoutSelIrc48M* IRC48MCLK.

**53.2.49.3.18 enum clock\_rtcout\_src\_k65f18\_t**

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

**53.2.49.3.19 enum clock\_usbhs\_slowclk\_src\_k65f18\_t**

Enumerator

*kClockUsbhsSlowClkSrcMcgIrClk* MCGIRCLK clock.  
*kClockUsbhsSlowClkSrcRtc32kHz* RTC 32kHz clock.

**53.2.49.3.20 enum sim\_usbsstby\_mode\_k65f18\_t**

Enumerator

*kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes  
*kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

**53.2.49.3.21 enum sim\_usbvstby\_mode\_k65f18\_t**

Enumerator

*kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes  
*kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes

**53.2.49.3.22 enum sim\_usbvout\_mode\_k65f18\_t**

Enumerator

*kSimUsbvout2\_733V* USB 3V regulator output voltage set to 2.733V.  
*kSimUsbvout3\_020V* USB 3V regulator output voltage set to 3.020V.  
*kSimUsbvout3\_074V* USB 3V regulator output voltage set to 3.074V.  
*kSimUsbvout3\_130V* USB 3V regulator output voltage set to 3.130V.  
*kSimUsbvout3\_188V* USB 3V regulator output voltage set to 3.188V.  
*kSimUsbvout3\_248V* USB 3V regulator output voltage set to 3.248V.  
*kSimUsbvout3\_310V* USB 3V regulator output voltage set to 3.310V.

## SIM HAL driver

### 53.2.49.3.23 enum sim\_adc\_pretrg\_sel\_k65f18\_t

Enumerator

- kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.
- kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.49.3.24 enum sim\_adc\_trg\_sel\_k65f18\_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.
- kSimAdcTrgSelTpm* TPMx channel 0 (A pretrigger) and channel 1 (B pretrigger)

### 53.2.49.3.25 enum sim\_uart\_rxsrc\_k65f18\_t

Enumerator

- kSimUartRxsrcPin* UARTx\_RX Pin.
- kSimUartRxsrcCmp0* CMP0.
- kSimUartRxsrcCmp1* CMP1.

### 53.2.49.3.26 enum sim\_uart\_txsrc\_k65f18\_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.

**53.2.49.3.27 enum sim\_ftm\_trg\_src\_k65f18\_t**

Enumerator

- kSimFtmTrgSrc0* FlexTimer x trigger y select 0.
- kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

**53.2.49.3.28 enum sim\_ftm\_clk\_sel\_k65f18\_t**

Enumerator

- kSimFtmClkSel0* FTM CLKIN0 pin.
- kSimFtmClkSel1* FTM CLKIN1 pin.

**53.2.49.3.29 enum sim\_ftm\_ch\_src\_k65f18\_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.

**53.2.49.3.30 enum sim\_ftm\_ch\_out\_src\_k65f18\_t**

Enumerator

- kSimFtmChOutSrc0* FlexTimer x channel y output source selection 0.
- kSimFtmChOutSrc1* FlexTimer x channel y output source selection 1.

**53.2.49.3.31 enum sim\_ftmflt\_sel\_k65f18\_t**

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

**53.2.49.3.32 enum sim\_tpm\_clk\_sel\_k65f18\_t**

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

## SIM HAL driver

### 53.2.49.3.33 enum sim\_tpm\_ch\_src\_k65f18\_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.

### 53.2.49.3.34 enum sim\_cmtuartpad\_strength\_k65f18\_t

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.

### 53.2.49.3.35 enum sim\_ptd7pad\_strength\_k65f18\_t

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

### 53.2.49.3.36 enum sim\_flexbus\_security\_level\_k65f18\_t

Enumerator

- kSimFbslLevel0* FlexBus security level 0.
- kSimFbslLevel1* FlexBus security level 1.
- kSimFbslLevel2* FlexBus security level 2.
- kSimFbslLevel3* FlexBus security level 3.

### 53.2.49.3.37 enum sim\_clock\_gate\_name\_k65f18\_t



## 53.2.50 K66F18 SIM HAL driver

### 53.2.50.1 Overview

The section describes the enumerations, macros and data structures for K66F18 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MK66F18.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_k66f18\\_t](#) {  
    [kClockWdogSrcLpoClk](#),  
    [kClockWdogSrcAltClk](#) }  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_k66f18\\_t](#) {  
    [kClockTraceSrcMcgoutClkDiv](#),  
    [kClockTraceSrcCoreClk](#) }  
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_k66f18\\_t](#) {  
    [kClockPortFilterSrcBusClk](#),  
    [kClockPortFilterSrcLpoClk](#) }  
*PORTx digital input filter clock source select.*
- enum [clock\\_tpm\\_src\\_k66f18\\_t](#) {  
    [kClockTpmSrcNone](#),  
    [kClockTpmSrcPlIFllSelDiv](#),  
    [kClockTpmSrcOsc0erClk](#),  
    [kClockTpmSrcMcgIrClk](#) }  
*TPM clock source select.*
- enum [clock\\_lptmr\\_src\\_k66f18\\_t](#) {  
    [kClockLptmrSrcMcgIrClk](#),  
    [kClockLptmrSrcLpoClk](#),  
    [kClockLptmrSrcEr32kClk](#),  
    [kClockLptmrSrcOsc0erClkUndiv](#) }  
*LPTMR clock source select.*
- enum [clock\\_lpuart\\_src\\_k66f18\\_t](#) {  
    [kClockLpuartSrcNone](#),  
    [kClockLpuartSrcPlIFllSel](#),  
    [kClockLpuartSrcOsc0erClk](#),

## SIM HAL driver

- `kClockLpuartSrcMcgIrClk` }

*SIM LPUART clock source.*
- enum `sim_lpuart_rxsrc_k66f18_t` {
  - `kSimLpuartRxsSrcPin`,
  - `kSimLpuartRxsSrcCmp0`,
  - `kSimLpuartRxsSrcCmp1` }
  - SIM LPUART RX source.*
- enum `sim_lpuart_txsrc_k66f18_t` {
  - `kSimLpuartTxsrcPin`,
  - `kSimLpuartTxsrcTpm1`,
  - `kSimLpuartTxsrcTpm2` }
  - SIM LPUART TX source.*
- enum `clock_time_src_k66f18_t` {
  - `kClockTimeSrcCoreSysClk`,
  - `kClockTimeSrcPllFllSel`,
  - `kClockTimeSrcOsc0erClk`,
  - `kClockTimeSrcExt` }
  - SIM timestamp clock source.*
- enum `clock_rmii_src_k66f18_t` {
  - `kClockRmiiSrcExtalClk`,
  - `kClockRmiiSrcExt` }
  - SIM RMII clock source.*
- enum `clock_usbfs_src_k66f18_t` {
  - `kClockUsbfsSrcExt`,
  - `kClockUsbfsSrcPllFllSel` }
  - SIM USB FS clock source.*
- enum `clock_flexcan_src_k66f18_t` {
  - `kClockFlexcanSrcOsc0erClk`,
  - `kClockFlexcanSrcBusClk` }
  - FLEXCAN clock source select.*
- enum `clock_sdhc_src_k66f18_t` {
  - `kClockSdhcSrcCoreSysClk`,
  - `kClockSdhcSrcPllFllSel`,
  - `kClockSdhcSrcOsc0erClk`,
  - `kClockSdhcSrcExt` }
  - SDHC clock source.*
- enum `clock_sai_src_k66f18_t` {
  - `kClockSaiSrcSysClk` = 0U,
  - `kClockSaiSrcOsc0erClk` = 1U,
  - `kClockSaiSrcPllFllSel` = 3U }
  - SAI clock source.*
- enum `clock_pllfl_sel_k66f18_t` {
  - `kClockPllFllSelFll` = 0U,
  - `kClockPllFllSelPll` = 1U,
  - `kClockPllFllSelUsb1pfd` = 2U,
  - `kClockPllFllSelIrc48M` = 3U }
  - SIM PLLFLLSEL clock source select.*

- enum `clock_er32k_src_k66f18_t` {  
`kClockEr32kSrcOsc0 = 0U,`  
`kClockEr32kSrcRtc = 2U,`  
`kClockEr32kSrcLpo = 3U }`  
*SIM external reference clock source select (OSC32KSEL).*
- enum `clock_clkout_src_k66f18_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_k66f18_t` {  
`kClockRtcoutSrc1Hz,`  
`kClockRtcoutSrc32kHz }`  
*SIM RTCCLKOUTSEL clock source select.*
- enum `clock_usbhs_slowclk_src_k66f18_t` {  
`kClockUsbhsSlowClkSrcMcgIrClk,`  
`kClockUsbhsSlowClkSrcRtc32kHz }`  
*SIM USBHS/USBPHY slow clock source select.*
- enum `sim_usbsstby_mode_k66f18_t` {  
`kSimUsbsstbyNoRegulator,`  
`kSimUsbsstbyWithRegulator }`  
*SIM USB voltage regulator in standby mode setting during stop modes.*
- enum `sim_usbvstby_mode_k66f18_t` {  
`kSimUsbvstbyNoRegulator,`  
`kSimUsbvstbyWithRegulator }`  
*SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.*
- enum `sim_usbvout_mode_k66f18_t` {  
`kSimUsbvout2_733V,`  
`kSimUsbvout3_020V,`  
`kSimUsbvout3_074V,`  
`kSimUsbvout3_130V,`  
`kSimUsbvout3_188V,`  
`kSimUsbvout3_248V,`  
`kSimUsbvout3_310V }`  
*SIM USB voltage regulator 3.3 output target.*
- enum `sim_adc_pretrg_sel_k66f18_t` {  
`kSimAdcPretrgselA,`  
`kSimAdcPretrgselB }`  
*SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_k66f18_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,
- kSimAdcTrgSelTpm = 15U }
- SIM ADCx trigger select.*
- enum `sim_uart_rxsrc_k66f18_t` {
  - kSimUartRxsrcPin,
  - kSimUartRxsrcCmp0,
  - kSimUartRxsrcCmp1 }
  - SIM UART receive data source select.*
  - enum `sim_uart_txsrc_k66f18_t` {
    - kSimUartTxsrcPin,
    - kSimUartTxsrcFtm1,
    - kSimUartTxsrcFtm2 }
    - SIM UART transmit data source select.*
    - enum `sim_ftm_trg_src_k66f18_t` {
      - kSimFtmTrgSrc0,
      - kSimFtmTrgSrc1 }
      - SIM FlexTimer x trigger y select.*
      - enum `sim_ftm_clk_sel_k66f18_t` {
        - kSimFtmClkSel0,
        - kSimFtmClkSel1 }
        - SIM FlexTimer external clock select.*
        - enum `sim_ftm_ch_src_k66f18_t` {
          - kSimFtmChSrc0,
          - kSimFtmChSrc1,
          - kSimFtmChSrc2,
          - kSimFtmChSrc3 }
          - SIM FlexTimer x channel y input capture source select.*
          - enum `sim_ftm_ch_out_src_k66f18_t` {
            - kSimFtmChOutSrc0,
            - kSimFtmChOutSrc1 }
            - SIM FlexTimer x channel y output source select.*
            - enum `sim_ftm_ft_sel_k66f18_t` {

- `kSimFtmFltSel0,`
  - `kSimFtmFltSel1 }`

*SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_k66f18_t` {
  - `kSimTpmClkSel0,`
  - `kSimTpmClkSel1 }`

*SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_k66f18_t` {
  - `kSimTpmChSrc0,`
  - `kSimTpmChSrc1,`
  - `kSimTpmChSrc2,`
  - `kSimTpmChSrc3 }`

*SIM Timer/PWM x channel y input capture source select.*
- enum `sim_cmtuartpad_strength_k66f18_t` {
  - `kSimCmtuartSinglePad,`
  - `kSimCmtuartDualPad }`

*SIM CMT/UART pad drive strength.*
- enum `sim_ptd7pad_strength_k66f18_t` {
  - `kSimPtd7padSinglePad,`
  - `kSimPtd7padDualPad }`

*SIM PTD7 pad drive strength.*
- enum `sim_flexbus_security_level_k66f18_t` {
  - `kSimFbslLevel0,`
  - `kSimFbslLevel1,`
  - `kSimFbslLevel2,`
  - `kSimFbslLevel3 }`

*SIM FlexBus security level.*
- enum `sim_clock_gate_name_k66f18_t`

*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.50.2 Macro Definition Documentation

53.2.50.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.50.3 Enumeration Type Documentation

53.2.50.3.1 enum `clock_wdog_src_k66f18_t`

Enumerator

*kClockWdogSrcLpoClk* LPO.

*kClockWdogSrcAltClk* Alternative clock, for this SOC it is Bus clock.

## SIM HAL driver

### 53.2.50.3.2 enum clock\_trace\_src\_k66f18\_t

Enumerator

*kClockTraceSrcMcgoutClkDiv* MCG out clock divided by the fractional divider configured by SIM\_CLKDIV4[TRACEFRAC, TRACEDIV].  
*kClockTraceSrcCoreClk* core clock

### 53.2.50.3.3 enum clock\_port\_filter\_src\_k66f18\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.  
*kClockPortFilterSrcLpoClk* LPO.

### 53.2.50.3.4 enum clock\_tpm\_src\_k66f18\_t

Enumerator

*kClockTpmSrcNone* clock disabled  
*kClockTpmSrcPllFllSelDiv* clock as selected by SOPT2[PLLFLLSEL] and divided by the fractional divider configured by SIM\_CLKDIV3[PLLFLLFRAC, PLLFLLDIV].  
*kClockTpmSrcOsc0erClk* OSCERCLK clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.50.3.5 enum clock\_lptmr\_src\_k66f18\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCGIRCLK.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClkUndiv* OSCERCLK\_UNDIV clock.

### 53.2.50.3.6 enum clock\_lpuart\_src\_k66f18\_t

Enumerator

*kClockLpuartSrcNone* Clock disabled.  
*kClockLpuartSrcPllFllSel* Clock as selected by SOPT2[PLLFLLSEL].  
*kClockLpuartSrcOsc0erClk* OSCERCLK.  
*kClockLpuartSrcMcgIrClk* MCGIRCLK.

**53.2.50.3.7 enum sim\_lpuart\_rxsrc\_k66f18\_t**

Enumerator

*kSimLpuartRxsrcPin* LPUARTx\_RX Pin.  
*kSimLpuartRxsrcCmp0* CMP0.  
*kSimLpuartRxsrcCmp1* CMP1.

**53.2.50.3.8 enum sim\_lpuart\_txsrc\_k66f18\_t**

Enumerator

*kSimLpuartTxsrcPin* UARTx\_TX Pin.  
*kSimLpuartTxsrcTpm1* UARTx\_TX pin modulated with TPM1 channel 0 output.  
*kSimLpuartTxsrcTpm2* UARTx\_TX pin modulated with TPM2 channel 0 output.

**53.2.50.3.9 enum clock\_time\_src\_k66f18\_t**

Enumerator

*kClockTimeSrcCoreSysClk* Core/system clock.  
*kClockTimeSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockTimeSrcOsc0erClk* OSCERCLK clock.  
*kClockTimeSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

**53.2.50.3.10 enum clock\_rmii\_src\_k66f18\_t**

Enumerator

*kClockRmiiSrcExtalClk* EXTAL Clock.  
*kClockRmiiSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

**53.2.50.3.11 enum clock\_usbfs\_src\_k66f18\_t**

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)  
*kClockUsbfsSrcPllFllSel* Clock divider USB FS clock.

**53.2.50.3.12 enum clock\_flexcan\_src\_k66f18\_t**

Enumerator

*kClockFlexcanSrcOsc0erClk* OSCERCLK.  
*kClockFlexcanSrcBusClk* Bus clock.

## SIM HAL driver

### 53.2.50.3.13 enum clock\_sdhc\_src\_k66f18\_t

Enumerator

*kClockSdhcSrcCoreSysClk* Core/system clock.  
*kClockSdhcSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockSdhcSrcOsc0erClk* OSCERCLK clock.  
*kClockSdhcSrcExt* External bypass clock (SDHC0\_CLKIN)

### 53.2.50.3.14 enum clock\_sai\_src\_k66f18\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllFllSel* MCGPLLFLCLK.

### 53.2.50.3.15 enum clock\_pllfl\_sel\_k66f18\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.  
*kClockPllFllSelUsb1pfd* USB1 PFD clock.  
*kClockPllFllSelIrc48M* IRC48MCLK.

### 53.2.50.3.16 enum clock\_er32k\_src\_k66f18\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.50.3.17 enum clock\_clkout\_src\_k66f18\_t

Enumerator

*kClockClkoutSelFlexbusClk* Flexbus clock.  
*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCGIRCLK.  
*kClockClkoutSelRtc* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSC0ERCLK.  
*kClockClkoutSelIrc48M* IRC48MCLK.



### 53.2.50.3.18 enum clock\_rtcout\_src\_k66f18\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

### 53.2.50.3.19 enum clock\_usbhs\_slowclk\_src\_k66f18\_t

Enumerator

*kClockUsbhsSlowClkSrcMcgIrClk* MCGIRCLK clock.  
*kClockUsbhsSlowClkSrcRtc32kHz* RTC 32kHz clock.

### 53.2.50.3.20 enum sim\_usbsstby\_mode\_k66f18\_t

Enumerator

*kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes  
*kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

### 53.2.50.3.21 enum sim\_usbvstby\_mode\_k66f18\_t

Enumerator

*kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes  
*kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes

### 53.2.50.3.22 enum sim\_usbvout\_mode\_k66f18\_t

Enumerator

*kSimUsbvout2\_733V* USB 3V regulator output voltage set to 2.733V.  
*kSimUsbvout3\_020V* USB 3V regulator output voltage set to 3.020V.  
*kSimUsbvout3\_074V* USB 3V regulator output voltage set to 3.074V.  
*kSimUsbvout3\_130V* USB 3V regulator output voltage set to 3.130V.  
*kSimUsbvout3\_188V* USB 3V regulator output voltage set to 3.188V.  
*kSimUsbvout3\_248V* USB 3V regulator output voltage set to 3.248V.  
*kSimUsbvout3\_310V* USB 3V regulator output voltage set to 3.310V.

## SIM HAL driver

### 53.2.50.3.23 enum sim\_adc\_pretrg\_sel\_k66f18\_t

Enumerator

- kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.
- kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.50.3.24 enum sim\_adc\_trg\_sel\_k66f18\_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.
- kSimAdcTrgSelTpm* TPMx channel 0 (A pretrigger) and channel 1 (B pretrigger)

### 53.2.50.3.25 enum sim\_uart\_rxsrc\_k66f18\_t

Enumerator

- kSimUartRxsrcPin* UARTx\_RX Pin.
- kSimUartRxsrcCmp0* CMP0.
- kSimUartRxsrcCmp1* CMP1.

### 53.2.50.3.26 enum sim\_uart\_txsrc\_k66f18\_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.

**53.2.50.3.27 enum sim\_ftm\_trg\_src\_k66f18\_t**

Enumerator

- kSimFtmTrgSrc0* FlexTimer x trigger y select 0.
- kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

**53.2.50.3.28 enum sim\_ftm\_clk\_sel\_k66f18\_t**

Enumerator

- kSimFtmClkSel0* FTM CLKIN0 pin.
- kSimFtmClkSel1* FTM CLKIN1 pin.

**53.2.50.3.29 enum sim\_ftm\_ch\_src\_k66f18\_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.

**53.2.50.3.30 enum sim\_ftm\_ch\_out\_src\_k66f18\_t**

Enumerator

- kSimFtmChOutSrc0* FlexTimer x channel y output source selection 0.
- kSimFtmChOutSrc1* FlexTimer x channel y output source selection 1.

**53.2.50.3.31 enum sim\_ftmflt\_sel\_k66f18\_t**

Enumerator

- kSimFtmFltSel0* FlexTimer x fault y select 0.
- kSimFtmFltSel1* FlexTimer x fault y select 1.

**53.2.50.3.32 enum sim\_tpm\_clk\_sel\_k66f18\_t**

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

## SIM HAL driver

### 53.2.50.3.33 enum sim\_tpm\_ch\_src\_k66f18\_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.

### 53.2.50.3.34 enum sim\_cmtuartpad\_strength\_k66f18\_t

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.

### 53.2.50.3.35 enum sim\_ptd7pad\_strength\_k66f18\_t

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

### 53.2.50.3.36 enum sim\_flexbus\_security\_level\_k66f18\_t

Enumerator

- kSimFbslLevel0* FlexBus security level 0.
- kSimFbslLevel1* FlexBus security level 1.
- kSimFbslLevel2* FlexBus security level 2.
- kSimFbslLevel3* FlexBus security level 3.

### 53.2.50.3.37 enum sim\_clock\_gate\_name\_k66f18\_t

## 53.2.51 KL43Z4 SIM HAL driver

### 53.2.51.1 Overview

The section describes the enumerations, macros and data structures for KL43Z4 SIM HAL driver. KL43Z4 SIM code is shared by KL17Z4, KL27Z4, KL33Z4 and KL43Z4.

#### Files

- file [fsl\\_sim\\_hal\\_MKL43Z4.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_cop\\_src\\_kl43z4\\_t](#) {  
  [kClockCopSrcLpoClk](#),  
  [kClockCopSrcMcgIrClk](#),  
  [kClockCopSrcOsc0erClk](#),  
  [kClockCopSrcBusClk](#) }  
*COP clock source selection.*
- enum [clock\\_er32k\\_src\\_kl43z4\\_t](#) {  
  [kClockEr32kSrcOsc0](#) = 0U,  
  [kClockEr32kSrcRtc](#) = 2U,  
  [kClockEr32kSrcLpo](#) = 3U }  
*SIM external reference clock source select (OSC32KSEL).*
- enum [clock\\_osc32kout\\_sel\\_kl43z4\\_t](#) {  
  [kClockOsc32koutNone](#) = 0U,  
  [kClockOsc32koutPte0](#) = 1U,  
  [kClockOsc32koutPte26](#) = 2U }  
*SIM external reference clock output pin select (OSC32KOUT).*
- enum [clock\\_lpuart\\_src\\_kl43z4\\_t](#) {  
  [kClockLpuartSrcNone](#),  
  [kClockLpuartSrcIrc48M](#),  
  [kClockLpuartSrcOsc0erClk](#),  
  [kClockLpuartSrcMcgIrClk](#) }  
*SIM LPUART clock source.*
- enum [clock\\_tpm\\_src\\_kl43z4\\_t](#) {  
  [kClockTpmSrcNone](#),  
  [kClockTpmSrcIrc48M](#),  
  [kClockTpmSrcOsc0erClk](#),  
  [kClockTpmSrcMcgIrClk](#) }

## SIM HAL driver

- SIM TPM clock source.*

  - enum `clock_flexio_src_kl43z4_t` {  
`kClockFlexioSrcNone`,  
`kClockFlexioSrcIrc48M`,  
`kClockFlexioSrcOsc0erClk`,  
`kClockFlexioSrcMcgIrClk` }
- FLEXIO clock source.*

  - enum `clock_lptmr_src_kl43z4_t` {  
`kClockLptmrSrcMcgIrClk`,  
`kClockLptmrSrcLpoClk`,  
`kClockLptmrSrcEr32kClk`,  
`kClockLptmrSrcOsc0erClk` }
- LPTMR clock source select.*

  - enum `clock_clkout_src_kl43z4_t` {  
`kClockClkoutSelFlashClk` = 2U,  
`kClockClkoutSelLpoClk` = 3U,  
`kClockClkoutSelMcgIrClk` = 4U,  
`kClockClkoutSelOsc0erClk` = 6U,  
`kClockClkoutSelIrc48M` = 7U }
- SIM CLKOUT\_SEL clock source select.*

  - enum `clock_rtcout_src_kl43z4_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrcOsc0erClk` }
- SIM RTCCLKOUTSEL clock source select.*

  - enum `sim_adc_pretrg_sel_kl43z4_t` {  
`kSimAdcPretrgselA`,  
`kSimAdcPretrgselB` }
- SIM ADCx pre-trigger select.*

  - enum `sim_adc_trg_sel_kl43z4_t` {  
`kSimAdcTrgSelExt` = 0U,  
`kSimAdcTrgSelComp0` = 1U,  
`kSimAdcTrgSelPit0` = 4U,  
`kSimAdcTrgSelPit1` = 5U,  
`kSimAdcTrgSelTpm0` = 8U,  
`kSimAdcTrgSelTpm1` = 9U,  
`kSimAdcTrgSelTpm2` = 10U,  
`kSimAdcTrgSelRtcAlarm` = 12U,  
`kSimAdcTrgSelRtcSec` = 13U,  
`kSimAdcTrgSelLptimer` = 14U }
- SIM ADCx trigger select.*

  - enum `sim_lpuart_rxsrc_kl43z4_t` {  
`kSimLpuartRxsrcPin`,  
`kSimLpuartRxsrcCmp0` }
- LPUART receive data source.*

  - enum `sim_lpuart_txsrc_kl43z4_t` {  
`kSimLpuartTxsrcPin`,  
`kSimLpuartTxsrcTpm1`,

- `kSimLpuartTxsrcTpm2` }  
*LPUART transmit data source.*
- enum `clock_sai_src_kl43z4_t` {  
`kClockSaiSrcSysClk` = 0U,  
`kClockSaiSrcOsc0erClk` = 1U,  
`kClockSaiSrcMcgIrClk` = 2U,  
`kClockSaiSrcIrc48M` = 3U }  
*SAI clock source.*
- enum `sim_tpm_clk_sel_kl43z4_t` {  
`kSimTpmClkSel0`,  
`kSimTpmClkSel1` }  
*SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_kl43z4_t` {  
`kSimTpmChSrc0`,  
`kSimTpmChSrc1`,  
`kSimTpmChSrc2`,  
`kSimTpmChSrc3` }  
*SIM Timer/PWM x channel y input capture source select.*
- enum `sim_clock_gate_name_kl43z4_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.51.2 Macro Definition Documentation

53.2.51.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.51.3 Enumeration Type Documentation

#### 53.2.51.3.1 enum `clock_cop_src_kl43z4_t`

Enumerator

*kClockCopSrcLpoClk* LPO clock 1K HZ.  
*kClockCopSrcMcgIrClk* MCG IRC Clock.  
*kClockCopSrcOsc0erClk* OSCER Clock.  
*kClockCopSrcBusClk* BUS clock.

#### 53.2.51.3.2 enum `clock_er32k_src_kl43z4_t`

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

## SIM HAL driver

### 53.2.51.3.3 enum clock\_osc32kout\_sel\_kl43z4\_t

Enumerator

*kClockOsc32koutNone* ERCLK32K is not output.  
*kClockOsc32koutPte0* ERCLK32K is output on PTE0.  
*kClockOsc32koutPte26* ERCLK32K is output on PTE26.

### 53.2.51.3.4 enum clock\_lpuart\_src\_kl43z4\_t

Enumerator

*kClockLpuartSrcNone* disabled  
*kClockLpuartSrcIrc48M* IRC48M.  
*kClockLpuartSrcOsc0erClk* OSCER clock.  
*kClockLpuartSrcMcgIrClk* MCGIR clock.

### 53.2.51.3.5 enum clock\_tpm\_src\_kl43z4\_t

Enumerator

*kClockTpmSrcNone* disabled  
*kClockTpmSrcIrc48M* IRC48M/MCGPCLK.  
*kClockTpmSrcOsc0erClk* OSCER clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.51.3.6 enum clock\_flexio\_src\_kl43z4\_t

Enumerator

*kClockFlexioSrcNone* Clock disabled.  
*kClockFlexioSrcIrc48M* MCGPCLK/IRC48M.  
*kClockFlexioSrcOsc0erClk* OSCERCLK.  
*kClockFlexioSrcMcgIrClk* MCGIRCLK.

### 53.2.51.3.7 enum clock\_lptmr\_src\_kl43z4\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.



### 53.2.51.3.8 enum clock\_clkout\_src\_kl43z4\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelOsc0erClk* OSCER clock.  
*kClockClkoutSelIrc48M* IRC48M clock.

### 53.2.51.3.9 enum clock\_rtcout\_src\_kl43z4\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrcOsc0erClk* OSCER clock.

### 53.2.51.3.10 enum sim\_adc\_pretrg\_sel\_kl43z4\_t

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.51.3.11 enum sim\_adc\_trg\_sel\_kl43z4\_t

Enumerator

*kSimAdcTrgselExt* External trigger.  
*kSimAdcTrgSelComp0* CMP0 output.  
*kSimAdcTrgSelPit0* PIT trigger 0.  
*kSimAdcTrgSelPit1* PIT trigger 1.  
*kSimAdcTrgSelTpm0* TPM0 overflow.  
*kSimAdcTrgSelTpm1* TPM1 overflow.  
*kSimAdcTrgSelTpm2* TPM2 overflow.  
*kSimAdcTrgSelRtcAlarm* RTC alarm.  
*kSimAdcTrgSelRtcSec* RTC seconds.  
*kSimAdcTrgSelLptimer* Low-power timer trigger.

### 53.2.51.3.12 enum sim\_lpuart\_rxsrc\_kl43z4\_t

Enumerator

*kSimLpuartRxsrcPin* LPUARTx\_RX Pin.  
*kSimLpuartRxsrcCmp0* CMP0.

## SIM HAL driver

### 53.2.51.3.13 enum sim\_lpuart\_txsrc\_kl43z4\_t

Enumerator

- kSimLpuartTxsrcPin* UARTx\_TX Pin.
- kSimLpuartTxsrcTpm1* UARTx\_TX pin modulated with TPM1 channel 0 output.
- kSimLpuartTxsrcTpm2* UARTx\_TX pin modulated with TPM2 channel 0 output.

### 53.2.51.3.14 enum clock\_sai\_src\_kl43z4\_t

Enumerator

- kClockSaiSrcSysClk* SYSCLK.
- kClockSaiSrcOsc0erClk* OSC0ERCLK.
- kClockSaiSrcMcgIrClk* MCGIRCLK.
- kClockSaiSrcIrc48M* MCGPCLK/IRC48M.

### 53.2.51.3.15 enum sim\_tpm\_clk\_sel\_kl43z4\_t

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

### 53.2.51.3.16 enum sim\_tpm\_ch\_src\_kl43z4\_t

Enumerator

- kSimTpmChSrc0* Channel y input capture source uses 0.
- kSimTpmChSrc1* Channel y input capture source uses 1.
- kSimTpmChSrc2* Channel y input capture source uses 2.
- kSimTpmChSrc3* Channel y input capture source uses 3.

### 53.2.51.3.17 enum sim\_clock\_gate\_name\_kl43z4\_t

## 53.2.52 KW01Z4 SIM HAL driver

### 53.2.52.1 Overview

The section describes the enumerations, macros and data structures for KW01Z4 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKW01Z4.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_cop\\_src\\_kw01z4\\_t](#) {  
  [kClockCopSrcLpoClk](#),  
  [kClockCopSrcAltClk](#) }  
*COP clock source select.*
- enum [clock\\_tpm\\_src\\_kw01z4\\_t](#) {  
  [kClockTpmSrcNone](#),  
  [kClockTpmSrcPllFllSel](#),  
  [kClockTpmSrcOsc0erClk](#),  
  [kClockTpmSrcMcgIrClk](#) }  
*TPM clock source select.*
- enum [clock\\_lptmr\\_src\\_kw01z4\\_t](#) {  
  [kClockLptmrSrcMcgIrClk](#),  
  [kClockLptmrSrcLpoClk](#),  
  [kClockLptmrSrcEr32kClk](#),  
  [kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_lpsci\\_src\\_kw01z4\\_t](#) {  
  [kClockLpsciSrcNone](#),  
  [kClockLpsciSrcPllFllSel](#),  
  [kClockLpsciSrcOsc0erClk](#),  
  [kClockLpsciSrcMcgIrClk](#) }  
*UART0 clock source select.*
- enum [clock\\_sai\\_src\\_kw01z4\\_t](#) {  
  [kClockSaiSrcSysClk](#) = 0U,  
  [kClockSaiSrcOsc0erClk](#) = 1U,  
  [kClockSaiSrcPllClk](#) = 3U }  
*SAI clock source.*

## SIM HAL driver

- enum `clock_pllfl1_sel_kw01z4_t` {  
`kClockPllFl1SelFl1`,  
`kClockPllFl1SelPll` }  
*SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_kw01z4_t` {  
`kClockEr32kSrcOsc0` = 0U,  
`kClockEr32kSrcReserved` = 1U,  
`kClockEr32kSrcRtc` = 2U,  
`kClockEr32kSrcLpo` = 3U }  
*SIM external reference clock source select (OSC32KSEL)*
- enum `clock_clkout_src_kw01z4_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_kw01z4_t` {  
`kClockRtcoutSrc1Hz`,  
`kClockRtcoutSrc32kHz` }  
*SIM RTCCLKOUTSEL clock source select.*
- enum `sim_adc_pretrg_sel_kw01z4_t` {  
`kSimAdcPretrgselA`,  
`kSimAdcPretrgselB` }  
*SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kw01z4_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_rxsrc_kw01z4_t` {  
`kSimUartRxsrcPin`,  
`kSimUartRxsrcCmp0` }  
*SIM UART receive data source select.*
- enum `sim_uart_txsrc_kw01z4_t` {  
`kSimUartTxsrcPin`,  
`kSimUartTxsrcTpm1`,  
`kSimUartTxsrcTpm2`,  
`kSimUartTxsrcReserved` }  
*SIM UART transmit data source select.*
- enum `sim_lpsci_rxsrc_kw01z4_t` {  
`kSimLpsciRxsrcPin`,  
`kSimLpsciRxsrcCmp0` }  
*SIM LPSCI receive data source select.*
- enum `sim_lpsci_txsrc_kw01z4_t` {  
`kSimLpsciTxsrcPin`,  
`kSimLpsciTxsrcTpm1`,  
`kSimLpsciTxsrcTpm2`,  
`kSimLpsciTxsrcReserved` }  
*SIM LPSCI transmit data source select.*
- enum `sim_tpm_clk_sel_kw01z4_t` {  
`kSimTpmClkSel0`,  
`kSimTpmClkSel1` }  
*SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_kw01z4_t` {  
`kSimTpmChSrc0`,  
`kSimTpmChSrc1`,  
`kSimTpmChSrc2`,  
`kSimTpmChSrc3` }  
*SIM Timer/PWM x channel y input capture source select.*
- enum `sim_clock_gate_name_kw01z4_t`  
*Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

## 53.2.52.2 Macro Definition Documentation

53.2.52.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

## 53.2.52.3 Enumeration Type Documentation

53.2.52.3.1 enum `clock_cop_src_kw01z4_t`

Enumerator

*kClockCopSrcLpoClk* LPO.

*kClockCopSrcAltClk* Alternative clock, for KW01Z4 it is Bus clock.

## SIM HAL driver

### 53.2.52.3.2 enum clock\_tpm\_src\_kw01z4\_t

Enumerator

*kClockTpmSrcNone* clock disabled  
*kClockTpmSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockTpmSrcOsc0erClk* OSCERCLK clock.  
*kClockTpmSrcMcgIrClk* MCGIR clock.

### 53.2.52.3.3 enum clock\_lptmr\_src\_kw01z4\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG out clock.  
*kClockLptmrSrcLpoClk* LPO clock.  
*kClockLptmrSrcEr32kClk* ERCLK32K clock.  
*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

### 53.2.52.3.4 enum clock\_lpsci\_src\_kw01z4\_t

Enumerator

*kClockLpsciSrcNone* clock disabled  
*kClockLpsciSrcPllFllSel* clock as selected by SOPT2[PLLFLSEL].  
*kClockLpsciSrcOsc0erClk* OSCERCLK clock.  
*kClockLpsciSrcMcgIrClk* MCGIR clock.

### 53.2.52.3.5 enum clock\_sai\_src\_kw01z4\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.52.3.6 enum clock\_pllfl\_sel\_kw01z4\_t

Enumerator

*kClockPllFllSelFll* Fll clock.  
*kClockPllFllSelPll* Pll0 clock.

**53.2.52.3.7 enum clock\_er32k\_src\_kw01z4\_t**

Enumerator

*kClockEr32kSrcOsc0* OSC 32k clock.  
*kClockEr32kSrcReserved* Reserved.  
*kClockEr32kSrcRtc* RTC 32k clock.  
*kClockEr32kSrcLpo* LPO clock.

**53.2.52.3.8 enum clock\_clkout\_src\_kw01z4\_t**

Enumerator

*kClockClkoutReserved* Reserved.  
*kClockClkoutReserved1* Reserved.  
*kClockClkoutBusClk* Bus clock.  
*kClockClkoutLpoClk* LPO clock.  
*kClockClkoutMcgIrClk* MCG ir clock.  
*kClockClkoutReserved2* Reserved.  
*kClockClkoutOsc0erClk* OSC0ER clock.  
*kClockClkoutReserved3* Reserved.

**53.2.52.3.9 enum clock\_rtcout\_src\_kw01z4\_t**

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32KHz clock

**53.2.52.3.10 enum sim\_adc\_pretrg\_sel\_kw01z4\_t**

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.  
*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

**53.2.52.3.11 enum sim\_adc\_trg\_sel\_kw01z4\_t**

Enumerator

*kSimAdcTrgSelExt* External trigger.  
*kSimAdcTrgSelComp0* CMP0 output.  
*kSimAdcTrgSelReserved* Reserved.

## SIM HAL driver

*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.

### 53.2.52.3.12 enum sim\_uart\_rxsrc\_kw01z4\_t

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.  
*kSimUartRxsrcCmp0* CMP0.

### 53.2.52.3.13 enum sim\_uart\_txsrc\_kw01z4\_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.

### 53.2.52.3.14 enum sim\_lpsci\_rxsrc\_kw01z4\_t

Enumerator

*kSimLpsciRxsrcPin* LPSCIx\_RX Pin.  
*kSimLpsciRxsrcCmp0* CMP0.

### 53.2.52.3.15 enum sim\_lpsci\_txsrc\_kw01z4\_t

Enumerator

*kSimLpsciTxsrcPin* LPSCIx\_TX Pin.



*kSimLpsciTxsrcTpm1* LPSCIx\_TX pin modulated with TPM1 channel 0 output.  
*kSimLpsciTxsrcTpm2* LPSCIx\_TX pin modulated with TPM2 channel 0 output.  
*kSimLpsciTxsrcReserved* Reserved.

#### 53.2.52.3.16 enum sim\_tpm\_clk\_sel\_kw01z4\_t

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.  
*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

#### 53.2.52.3.17 enum sim\_tpm\_ch\_src\_kw01z4\_t

Enumerator

*kSimTpmChSrc0* TPMx\_CH0 signal.  
*kSimTpmChSrc1* CMP0 output.  
*kSimTpmChSrc2* Reserved.  
*kSimTpmChSrc3* USB start of frame pulse.

#### 53.2.52.3.18 enum sim\_clock\_gate\_name\_kw01z4\_t

## SIM HAL driver

### 53.2.53 KW21D5 SIM HAL driver

#### 53.2.53.1 Overview

The section describes the enumerations, macros and data structures for KW21D5 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKW21D5.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_kw21d5\\_t](#)  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_kw21d5\\_t](#) {  
[kClockTraceSrcMcgoutClk](#),  
[kClockTraceSrcCoreClk](#) }  
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_kw21d5\\_t](#) {  
[kClockPortFilterSrcBusClk](#),  
[kClockPortFilterSrcLpoClk](#) }  
*PORTx digital input filter clock source select.*
- enum [clock\\_lptmr\\_src\\_kw21d5\\_t](#) {  
[kClockLptmrSrcMcgIrClk](#),  
[kClockLptmrSrcLpoClk](#),  
[kClockLptmrSrcEr32kClk](#),  
[kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_time\\_src\\_kw21d5\\_t](#) {  
[kClockTimeSrcCoreSysClk](#),  
[kClockTimeSrcPIIFllSel](#),  
[kClockTimeSrcOsc0erClk](#),  
[kClockTimeSrcExt](#) }  
*SIM timestamp clock source.*
- enum [clock\\_usbfs\\_src\\_kw21d5\\_t](#) {  
[kClockUsbfsSrcExt](#),  
[kClockUsbfsSrcPIIFllSel](#) }  
*SIM USB FS clock source.*
- enum [clock\\_sai\\_src\\_kw21d5\\_t](#) {  
[kClockSaiSrcSysClk](#) = 0U,  
[kClockSaiSrcOsc0erClk](#) = 1U,

- `kClockSaiSrcPllClk = 3U }`
  - SAI clock source.*
- enum `clock_pllfl1_sel_kw21d5_t` {
  - `kClockPllFl1SelFl1 = 0U,`
  - `kClockPllFl1SelPll = 1U }`
  - SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_kw21d5_t` {
  - `kClockEr32kSrcOsc0 = 0U,`
  - `kClockEr32kSrcRtc = 2U,`
  - `kClockEr32kSrcLpo = 3U }`
  - SIM external reference clock source select (OSC32KSEL).*
- enum `clock_clkout_src_kw21d5_t` {
  - `kClockClkoutSelFlashClk = 2U,`
  - `kClockClkoutSelLpoClk = 3U,`
  - `kClockClkoutSelMcgIrClk = 4U,`
  - `kClockClkoutSelRtc32kClk = 5U,`
  - `kClockClkoutSelOsc0erClk = 6U }`
  - SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_kw21d5_t` {
  - `kClockRtcoutSrc1Hz,`
  - `kClockRtcoutSrc32kHz }`
  - SIM RTCCLKOUTSEL clock source select.*
- enum `sim_usbsstby_mode_kw21d5_t` {
  - `kSimUsbsstbyNoRegulator,`
  - `kSimUsbsstbyWithRegulator }`
  - SIM USB voltage regulator in standby mode setting during stop modes.*
- enum `sim_usbvstby_mode_kw21d5_t` {
  - `kSimUsbvstbyNoRegulator,`
  - `kSimUsbvstbyWithRegulator }`
  - SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.*
- enum `sim_adc_pretrg_sel_kw21d5_t` {
  - `kSimAdcPretrgselA,`
  - `kSimAdcPretrgselB }`
  - SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kw21d5_t` {
  - `kSimAdcTrgselExt = 0U,`
  - `kSimAdcTrgSelHighSpeedComp0 = 1U,`
  - `kSimAdcTrgSelHighSpeedComp1 = 2U,`
  - `kSimAdcTrgSelPit0 = 4U,`
  - `kSimAdcTrgSelPit1 = 5U,`
  - `kSimAdcTrgSelPit2 = 6U,`
  - `kSimAdcTrgSelPit3 = 7U,`
  - `kSimAdcTrgSelFtm0 = 8U,`
  - `kSimAdcTrgSelFtm1 = 9U,`
  - `kSimAdcTrgSelFtm2 = 10U,`
  - `kSimAdcTrgSelRtcAlarm = 12U,`
  - `kSimAdcTrgSelRtcSec = 13U,`

## SIM HAL driver

- `kSimAdcTrgSellptimer = 14U }`
  - SIM ADCx trigger select.*
- enum `sim_uart_rxsrc_kw21d5_t` {
  - `kSimUartRxsrcPin,`
  - `kSimUartRxsrcCmp0,`
  - `kSimUartRxsrcCmp1 }`
  - SIM UART receive data source select.*
- enum `sim_uart_txsrc_kw21d5_t` {
  - `kSimUartTxsrcPin,`
  - `kSimUartTxsrcFtm1,`
  - `kSimUartTxsrcFtm2 }`
  - SIM UART transmit data source select.*
- enum `sim_ftm_trg_src_kw21d5_t` {
  - `kSimFtmTrgSrc0,`
  - `kSimFtmTrgSrc1 }`
  - SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_kw21d5_t` {
  - `kSimFtmClkSel0,`
  - `kSimFtmClkSel1 }`
  - SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_kw21d5_t` {
  - `kSimFtmChSrc0,`
  - `kSimFtmChSrc1,`
  - `kSimFtmChSrc2,`
  - `kSimFtmChSrc3 }`
  - SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftmflt_sel_kw21d5_t` {
  - `kSimFtmFltSel0,`
  - `kSimFtmFltSel1 }`
  - SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_kw21d5_t` {
  - `kSimTpmClkSel0,`
  - `kSimTpmClkSel1 }`
  - SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_kw21d5_t` {
  - `kSimTpmChSrc0,`
  - `kSimTpmChSrc1 }`
  - SIM Timer/PWM x channel y input capture source select.*
- enum `sim_cmtuartpad_strenght_kw21d5_t` {
  - `kSimCmtuartSinglePad,`
  - `kSimCmtuartDualPad }`
  - SIM CMT/UART pad drive strength.*
- enum `sim_ptd7pad_strenght_kw21d5_t` {
  - `kSimPtd7padSinglePad,`
  - `kSimPtd7padDualPad }`
  - SIM PTD7 pad drive strength.*
- enum `sim_clock_gate_name_kw21d5_t`
  - Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.53.2 Macro Definition Documentation

53.2.53.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.53.3 Enumeration Type Documentation

#### 53.2.53.3.1 enum clock\_trace\_src\_kw21d5\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

#### 53.2.53.3.2 enum clock\_port\_filter\_src\_kw21d5\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.

*kClockPortFilterSrcLpoClk* LPO.

#### 53.2.53.3.3 enum clock\_lptmr\_src\_kw21d5\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG IRC clock.

*kClockLptmrSrcLpoClk* LPO clock.

*kClockLptmrSrcEr32kClk* ERCLK32K clock.

*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

#### 53.2.53.3.4 enum clock\_time\_src\_kw21d5\_t

Enumerator

*kClockTimeSrcCoreSysClk* Core/system clock.

*kClockTimeSrcPllFllSel* clock as selected by SOPT2[PLLFLLSSEL].

*kClockTimeSrcOsc0erClk* OSCERCLK clock.

*kClockTimeSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

#### 53.2.53.3.5 enum clock\_usbfs\_src\_kw21d5\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)

*kClockUsbfsSrcPllFllSel* Clock divider USB FS clock.

## SIM HAL driver

### 53.2.53.3.6 enum clock\_sai\_src\_kw21d5\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.53.3.7 enum clock\_pllfl\_sel\_kw21d5\_t

Enumerator

*kClockPllFlSelFl* Fll clock.  
*kClockPllFlSelPll* Pll0 clock.

### 53.2.53.3.8 enum clock\_er32k\_src\_kw21d5\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.53.3.9 enum clock\_clkout\_src\_kw21d5\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelRtc32kClk* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSCERCLK0 clock.

### 53.2.53.3.10 enum clock\_rtcout\_src\_kw21d5\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

**53.2.53.3.11 enum sim\_usbsstby\_mode\_kw21d5\_t**

Enumerator

*kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes*kSimUsbsstbyWithRegulator* regulator in standby during Stop modes**53.2.53.3.12 enum sim\_usbvstby\_mode\_kw21d5\_t**

Enumerator

*kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes*kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes**53.2.53.3.13 enum sim\_adc\_pretrg\_sel\_kw21d5\_t**

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.**53.2.53.3.14 enum sim\_adc\_trg\_sel\_kw21d5\_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.**53.2.53.3.15 enum sim\_uart\_rxsrc\_kw21d5\_t**

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.

## SIM HAL driver

*kSimUartRxsrcCmp0* CMP0.

*kSimUartRxsrcCmp1* CMP1.

### 53.2.53.3.16 enum sim\_uart\_txsrc\_kw21d5\_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.

### 53.2.53.3.17 enum sim\_ftm\_trg\_src\_kw21d5\_t

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.

*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

### 53.2.53.3.18 enum sim\_ftm\_clk\_sel\_kw21d5\_t

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.

*kSimFtmClkSel1* FTM CLKIN1 pin.

### 53.2.53.3.19 enum sim\_ftm\_ch\_src\_kw21d5\_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.

### 53.2.53.3.20 enum sim\_ftmflt\_sel\_kw21d5\_t

Enumerator

*kSimFtmFltSel0* FlexTimer x fault y select 0.

*kSimFtmFltSel1* FlexTimer x fault y select 1.



**53.2.53.3.21 enum sim\_tpm\_clk\_sel\_kw21d5\_t**

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

**53.2.53.3.22 enum sim\_tpm\_ch\_src\_kw21d5\_t**

Enumerator

- kSimTpmChSrc0* TPMx\_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

**53.2.53.3.23 enum sim\_cmtuartpad\_streng\_h\_kw21d5\_t**

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.

**53.2.53.3.24 enum sim\_ptd7pad\_streng\_h\_kw21d5\_t**

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

**53.2.53.3.25 enum sim\_clock\_gate\_name\_kw21d5\_t**

## SIM HAL driver

### 53.2.54 KW22D5 SIM HAL driver

#### 53.2.54.1 Overview

The section describes the enumerations, macros and data structures for KW22D5 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKW22D5.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_kw22d5\\_t](#)  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_kw22d5\\_t](#) {  
[kClockTraceSrcMcgoutClk](#),  
[kClockTraceSrcCoreClk](#) }  
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_kw22d5\\_t](#) {  
[kClockPortFilterSrcBusClk](#),  
[kClockPortFilterSrcLpoClk](#) }  
*PORTx digital input filter clock source select.*
- enum [clock\\_lptmr\\_src\\_kw22d5\\_t](#) {  
[kClockLptmrSrcMcgIrClk](#),  
[kClockLptmrSrcLpoClk](#),  
[kClockLptmrSrcEr32kClk](#),  
[kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_time\\_src\\_kw22d5\\_t](#) {  
[kClockTimeSrcCoreSysClk](#),  
[kClockTimeSrcPIIFllSel](#),  
[kClockTimeSrcOsc0erClk](#),  
[kClockTimeSrcExt](#) }  
*SIM timestamp clock source.*
- enum [clock\\_usbfs\\_src\\_kw22d5\\_t](#) {  
[kClockUsbfsSrcExt](#),  
[kClockUsbfsSrcPIIFllSel](#) }  
*SIM USB FS clock source.*
- enum [clock\\_sai\\_src\\_kw22d5\\_t](#) {  
[kClockSaiSrcSysClk](#) = 0U,  
[kClockSaiSrcOsc0erClk](#) = 1U,

- `kClockSaiSrcPllClk = 3U }`
  - SAI clock source.*
- enum `clock_pllfl1_sel_kw22d5_t` {
  - `kClockPllFl1SelFl1 = 0U,`
  - `kClockPllFl1SelPll = 1U }`
  - SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_kw22d5_t` {
  - `kClockEr32kSrcOsc0 = 0U,`
  - `kClockEr32kSrcRtc = 2U,`
  - `kClockEr32kSrcLpo = 3U }`
  - SIM external reference clock source select (OSC32KSEL).*
- enum `clock_clkout_src_kw22d5_t` {
  - `kClockClkoutSelFlashClk = 2U,`
  - `kClockClkoutSelLpoClk = 3U,`
  - `kClockClkoutSelMcgIrClk = 4U,`
  - `kClockClkoutSelRtc32kClk = 5U,`
  - `kClockClkoutSelOsc0erClk = 6U }`
  - SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_kw22d5_t` {
  - `kClockRtcoutSrc1Hz,`
  - `kClockRtcoutSrc32kHz }`
  - SIM RTCCLKOUTSEL clock source select.*
- enum `sim_usbsstby_mode_kw22d5_t` {
  - `kSimUsbsstbyNoRegulator,`
  - `kSimUsbsstbyWithRegulator }`
  - SIM USB voltage regulator in standby mode setting during stop modes.*
- enum `sim_usbvstby_mode_kw22d5_t` {
  - `kSimUsbvstbyNoRegulator,`
  - `kSimUsbvstbyWithRegulator }`
  - SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.*
- enum `sim_adc_pretrg_sel_kw22d5_t` {
  - `kSimAdcPretrgselA,`
  - `kSimAdcPretrgselB }`
  - SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kw22d5_t` {
  - `kSimAdcTrgselExt = 0U,`
  - `kSimAdcTrgSelHighSpeedComp0 = 1U,`
  - `kSimAdcTrgSelHighSpeedComp1 = 2U,`
  - `kSimAdcTrgSelPit0 = 4U,`
  - `kSimAdcTrgSelPit1 = 5U,`
  - `kSimAdcTrgSelPit2 = 6U,`
  - `kSimAdcTrgSelPit3 = 7U,`
  - `kSimAdcTrgSelFtm0 = 8U,`
  - `kSimAdcTrgSelFtm1 = 9U,`
  - `kSimAdcTrgSelFtm2 = 10U,`
  - `kSimAdcTrgSelRtcAlarm = 12U,`
  - `kSimAdcTrgSelRtcSec = 13U,`

## SIM HAL driver

- `kSimAdcTrgSellptimer = 14U }`
  - SIM ADCx trigger select.*
- enum `sim_uart_rxsrc_kw22d5_t` {
  - `kSimUartRxsrcPin,`
  - `kSimUartRxsrcCmp0,`
  - `kSimUartRxsrcCmp1 }`
  - SIM UART receive data source select.*
- enum `sim_uart_txsrc_kw22d5_t` {
  - `kSimUartTxsrcPin,`
  - `kSimUartTxsrcFtm1,`
  - `kSimUartTxsrcFtm2 }`
  - SIM UART transmit data source select.*
- enum `sim_ftm_trg_src_kw22d5_t` {
  - `kSimFtmTrgSrc0,`
  - `kSimFtmTrgSrc1 }`
  - SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_kw22d5_t` {
  - `kSimFtmClkSel0,`
  - `kSimFtmClkSel1 }`
  - SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_kw22d5_t` {
  - `kSimFtmChSrc0,`
  - `kSimFtmChSrc1,`
  - `kSimFtmChSrc2,`
  - `kSimFtmChSrc3 }`
  - SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftmflt_sel_kw22d5_t` {
  - `kSimFtmFltSel0,`
  - `kSimFtmFltSel1 }`
  - SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_kw22d5_t` {
  - `kSimTpmClkSel0,`
  - `kSimTpmClkSel1 }`
  - SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_kw22d5_t` {
  - `kSimTpmChSrc0,`
  - `kSimTpmChSrc1 }`
  - SIM Timer/PWM x channel y input capture source select.*
- enum `sim_cmtuartpad_strenght_kw22d5_t` {
  - `kSimCmtuartSinglePad,`
  - `kSimCmtuartDualPad }`
  - SIM CMT/UART pad drive strength.*
- enum `sim_ptd7pad_strenght_kw22d5_t` {
  - `kSimPtd7padSinglePad,`
  - `kSimPtd7padDualPad }`
  - SIM PTD7 pad drive strength.*
- enum `sim_clock_gate_name_kw22d5_t`
  - Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.54.2 Macro Definition Documentation

53.2.54.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.54.3 Enumeration Type Documentation

#### 53.2.54.3.1 enum clock\_trace\_src\_kw22d5\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

#### 53.2.54.3.2 enum clock\_port\_filter\_src\_kw22d5\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.

*kClockPortFilterSrcLpoClk* LPO.

#### 53.2.54.3.3 enum clock\_lptmr\_src\_kw22d5\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG IRC clock.

*kClockLptmrSrcLpoClk* LPO clock.

*kClockLptmrSrcEr32kClk* ERCLK32K clock.

*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

#### 53.2.54.3.4 enum clock\_time\_src\_kw22d5\_t

Enumerator

*kClockTimeSrcCoreSysClk* Core/system clock.

*kClockTimeSrcPllFllSel* clock as selected by SOPT2[PLLFLLSSEL].

*kClockTimeSrcOsc0erClk* OSCERCLK clock.

*kClockTimeSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

#### 53.2.54.3.5 enum clock\_usbfs\_src\_kw22d5\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)

*kClockUsbfsSrcPllFllSel* Clock divider USB FS clock.

## SIM HAL driver

### 53.2.54.3.6 enum clock\_sai\_src\_kw22d5\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.54.3.7 enum clock\_pllfl\_sel\_kw22d5\_t

Enumerator

*kClockPllFlSelFl* Fll clock.  
*kClockPllFlSelPll* Pll0 clock.

### 53.2.54.3.8 enum clock\_er32k\_src\_kw22d5\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.54.3.9 enum clock\_clkout\_src\_kw22d5\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelRtc32kClk* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSCERCLK0 clock.

### 53.2.54.3.10 enum clock\_rtcout\_src\_kw22d5\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

**53.2.54.3.11 enum sim\_usbsstby\_mode\_kw22d5\_t**

Enumerator

*kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes*kSimUsbsstbyWithRegulator* regulator in standby during Stop modes**53.2.54.3.12 enum sim\_usbvstby\_mode\_kw22d5\_t**

Enumerator

*kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes*kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes**53.2.54.3.13 enum sim\_adc\_pretrg\_sel\_kw22d5\_t**

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.**53.2.54.3.14 enum sim\_adc\_trg\_sel\_kw22d5\_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.**53.2.54.3.15 enum sim\_uart\_rxsrc\_kw22d5\_t**

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.

## SIM HAL driver

*kSimUartRxsrcCmp0* CMP0.

*kSimUartRxsrcCmp1* CMP1.

### 53.2.54.3.16 enum sim\_uart\_txsrc\_kw22d5\_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.

### 53.2.54.3.17 enum sim\_ftm\_trg\_src\_kw22d5\_t

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.

*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

### 53.2.54.3.18 enum sim\_ftm\_clk\_sel\_kw22d5\_t

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.

*kSimFtmClkSel1* FTM CLKIN1 pin.

### 53.2.54.3.19 enum sim\_ftm\_ch\_src\_kw22d5\_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.

### 53.2.54.3.20 enum sim\_ftmflt\_sel\_kw22d5\_t

Enumerator

*kSimFtmFltSel0* FlexTimer x fault y select 0.

*kSimFtmFltSel1* FlexTimer x fault y select 1.



**53.2.54.3.21 enum sim\_tpm\_clk\_sel\_kw22d5\_t**

Enumerator

- kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.
- kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.

**53.2.54.3.22 enum sim\_tpm\_ch\_src\_kw22d5\_t**

Enumerator

- kSimTpmChSrc0* TPMx\_CH0 signal.
- kSimTpmChSrc1* CMP0 output.

**53.2.54.3.23 enum sim\_cmtuartpad\_streng\_h\_kw22d5\_t**

Enumerator

- kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.
- kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.

**53.2.54.3.24 enum sim\_ptd7pad\_streng\_h\_kw22d5\_t**

Enumerator

- kSimPtd7padSinglePad* Single-pad drive strength for PTD7.
- kSimPtd7padDualPad* Dual-pad drive strength for PTD7.

**53.2.54.3.25 enum sim\_clock\_gate\_name\_kw22d5\_t**

## SIM HAL driver

### 53.2.55 KW24D5 SIM HAL driver

#### 53.2.55.1 Overview

The section describes the enumerations, macros and data structures for KW24D5 SIM HAL driver.

#### Files

- file [fsl\\_sim\\_hal\\_MKW24D5.h](#)

#### Macros

- #define [FSL\\_SIM\\_SCGC\\_BIT](#)(SCGCx, n) (((SCGCx-1U)<<5U) + n)  
*SIM SCGC bit index.*

#### Enumerations

- enum [clock\\_wdog\\_src\\_kw24d5\\_t](#)  
*WDOG clock source select.*
- enum [clock\\_trace\\_src\\_kw24d5\\_t](#) {  
[kClockTraceSrcMcgoutClk](#),  
[kClockTraceSrcCoreClk](#) }  
*Debug trace clock source select.*
- enum [clock\\_port\\_filter\\_src\\_kw24d5\\_t](#) {  
[kClockPortFilterSrcBusClk](#),  
[kClockPortFilterSrcLpoClk](#) }  
*PORTx digital input filter clock source select.*
- enum [clock\\_lptmr\\_src\\_kw24d5\\_t](#) {  
[kClockLptmrSrcMcgIrClk](#),  
[kClockLptmrSrcLpoClk](#),  
[kClockLptmrSrcEr32kClk](#),  
[kClockLptmrSrcOsc0erClk](#) }  
*LPTMR clock source select.*
- enum [clock\\_time\\_src\\_kw24d5\\_t](#) {  
[kClockTimeSrcCoreSysClk](#),  
[kClockTimeSrcPIIFllSel](#),  
[kClockTimeSrcOsc0erClk](#),  
[kClockTimeSrcExt](#) }  
*SIM timestamp clock source.*
- enum [clock\\_usbfs\\_src\\_kw24d5\\_t](#) {  
[kClockUsbfsSrcExt](#),  
[kClockUsbfsSrcPIIFllSel](#) }  
*SIM USB FS clock source.*
- enum [clock\\_sai\\_src\\_kw24d5\\_t](#) {  
[kClockSaiSrcSysClk](#) = 0U,  
[kClockSaiSrcOsc0erClk](#) = 1U,

- `kClockSaiSrcPllClk = 3U }`
  - SAI clock source.*
- enum `clock_pllfl1_sel_kw24d5_t` {
  - `kClockPllFl1SelFl1 = 0U,`
  - `kClockPllFl1SelPll = 1U }`
  - SIM PLLFLLSEL clock source select.*
- enum `clock_er32k_src_kw24d5_t` {
  - `kClockEr32kSrcOsc0 = 0U,`
  - `kClockEr32kSrcRtc = 2U,`
  - `kClockEr32kSrcLpo = 3U }`
  - SIM external reference clock source select (OSC32KSEL).*
- enum `clock_clkout_src_kw24d5_t` {
  - `kClockClkoutSelFlashClk = 2U,`
  - `kClockClkoutSelLpoClk = 3U,`
  - `kClockClkoutSelMcgIrClk = 4U,`
  - `kClockClkoutSelRtc32kClk = 5U,`
  - `kClockClkoutSelOsc0erClk = 6U }`
  - SIM CLKOUT\_SEL clock source select.*
- enum `clock_rtcout_src_kw24d5_t` {
  - `kClockRtcoutSrc1Hz,`
  - `kClockRtcoutSrc32kHz }`
  - SIM RTCCLKOUTSEL clock source select.*
- enum `sim_usbsstby_mode_kw24d5_t` {
  - `kSimUsbsstbyNoRegulator,`
  - `kSimUsbsstbyWithRegulator }`
  - SIM USB voltage regulator in standby mode setting during stop modes.*
- enum `sim_usbvstby_mode_kw24d5_t` {
  - `kSimUsbvstbyNoRegulator,`
  - `kSimUsbvstbyWithRegulator }`
  - SIM USB voltage regulator in standby mode setting during VLPR and VLPW modes.*
- enum `sim_adc_pretrg_sel_kw24d5_t` {
  - `kSimAdcPretrgselA,`
  - `kSimAdcPretrgselB }`
  - SIM ADCx pre-trigger select.*
- enum `sim_adc_trg_sel_kw24d5_t` {
  - `kSimAdcTrgselExt = 0U,`
  - `kSimAdcTrgSelHighSpeedComp0 = 1U,`
  - `kSimAdcTrgSelHighSpeedComp1 = 2U,`
  - `kSimAdcTrgSelPit0 = 4U,`
  - `kSimAdcTrgSelPit1 = 5U,`
  - `kSimAdcTrgSelPit2 = 6U,`
  - `kSimAdcTrgSelPit3 = 7U,`
  - `kSimAdcTrgSelFtm0 = 8U,`
  - `kSimAdcTrgSelFtm1 = 9U,`
  - `kSimAdcTrgSelFtm2 = 10U,`
  - `kSimAdcTrgSelRtcAlarm = 12U,`
  - `kSimAdcTrgSelRtcSec = 13U,`

## SIM HAL driver

- `kSimAdcTrgSellptimer = 14U }`
  - SIM ADCx trigger select.*
- enum `sim_uart_rxsrc_kw24d5_t` {
  - `kSimUartRxsrcPin,`
  - `kSimUartRxsrcCmp0,`
  - `kSimUartRxsrcCmp1 }`
  - SIM UART receive data source select.*
- enum `sim_uart_txsrc_kw24d5_t` {
  - `kSimUartTxsrcPin,`
  - `kSimUartTxsrcFtm1,`
  - `kSimUartTxsrcFtm2 }`
  - SIM UART transmit data source select.*
- enum `sim_ftm_trg_src_kw24d5_t` {
  - `kSimFtmTrgSrc0,`
  - `kSimFtmTrgSrc1 }`
  - SIM FlexTimer x trigger y select.*
- enum `sim_ftm_clk_sel_kw24d5_t` {
  - `kSimFtmClkSel0,`
  - `kSimFtmClkSel1 }`
  - SIM FlexTimer external clock select.*
- enum `sim_ftm_ch_src_kw24d5_t` {
  - `kSimFtmChSrc0,`
  - `kSimFtmChSrc1,`
  - `kSimFtmChSrc2,`
  - `kSimFtmChSrc3 }`
  - SIM FlexTimer x channel y input capture source select.*
- enum `sim_ftmflt_sel_kw24d5_t` {
  - `kSimFtmFltSel0,`
  - `kSimFtmFltSel1 }`
  - SIM FlexTimer x Fault y select.*
- enum `sim_tpm_clk_sel_kw24d5_t` {
  - `kSimTpmClkSel0,`
  - `kSimTpmClkSel1 }`
  - SIM Timer/PWM external clock select.*
- enum `sim_tpm_ch_src_kw24d5_t` {
  - `kSimTpmChSrc0,`
  - `kSimTpmChSrc1 }`
  - SIM Timer/PWM x channel y input capture source select.*
- enum `sim_cmtuartpad_strenght_kw24d5_t` {
  - `kSimCmtuartSinglePad,`
  - `kSimCmtuartDualPad }`
  - SIM CMT/UART pad drive strength.*
- enum `sim_ptd7pad_strenght_kw24d5_t` {
  - `kSimPtd7padSinglePad,`
  - `kSimPtd7padDualPad }`
  - SIM PTD7 pad drive strength.*
- enum `sim_clock_gate_name_kw24d5_t`
  - Clock gate name used for SIM\_HAL\_EnableClock/SIM\_HAL\_DisableClock.*

### 53.2.55.2 Macro Definition Documentation

53.2.55.2.1 `#define FSL_SIM_SCGC_BIT( SCGCx, n ) (((SCGCx-1U)<<5U) + n)`

### 53.2.55.3 Enumeration Type Documentation

#### 53.2.55.3.1 enum clock\_trace\_src\_kw24d5\_t

Enumerator

*kClockTraceSrcMcgoutClk* MCG out clock.

*kClockTraceSrcCoreClk* core clock

#### 53.2.55.3.2 enum clock\_port\_filter\_src\_kw24d5\_t

Enumerator

*kClockPortFilterSrcBusClk* Bus clock.

*kClockPortFilterSrcLpoClk* LPO.

#### 53.2.55.3.3 enum clock\_lptmr\_src\_kw24d5\_t

Enumerator

*kClockLptmrSrcMcgIrClk* MCG IRC clock.

*kClockLptmrSrcLpoClk* LPO clock.

*kClockLptmrSrcEr32kClk* ERCLK32K clock.

*kClockLptmrSrcOsc0erClk* OSCERCLK clock.

#### 53.2.55.3.4 enum clock\_time\_src\_kw24d5\_t

Enumerator

*kClockTimeSrcCoreSysClk* Core/system clock.

*kClockTimeSrcPllFllSel* clock as selected by SOPT2[PLLFLLSEL].

*kClockTimeSrcOsc0erClk* OSCERCLK clock.

*kClockTimeSrcExt* ENET 1588 clock in (ENET\_1588\_CLKIN)

#### 53.2.55.3.5 enum clock\_usbfs\_src\_kw24d5\_t

Enumerator

*kClockUsbfsSrcExt* External bypass clock (USB\_CLKIN)

*kClockUsbfsSrcPllFllSel* Clock divider USB FS clock.

## SIM HAL driver

### 53.2.55.3.6 enum clock\_sai\_src\_kw24d5\_t

Enumerator

*kClockSaiSrcSysClk* SYSCLK.  
*kClockSaiSrcOsc0erClk* OSC0ERCLK.  
*kClockSaiSrcPllClk* MCGPLLCLK.

### 53.2.55.3.7 enum clock\_pllfl\_sel\_kw24d5\_t

Enumerator

*kClockPllFlSelFl* Fll clock.  
*kClockPllFlSelPll* Pll0 clock.

### 53.2.55.3.8 enum clock\_er32k\_src\_kw24d5\_t

Enumerator

*kClockEr32kSrcOsc0* OSC0 clock (OSC032KCLK).  
*kClockEr32kSrcRtc* RTC 32k clock .  
*kClockEr32kSrcLpo* LPO clock.

### 53.2.55.3.9 enum clock\_clkout\_src\_kw24d5\_t

Enumerator

*kClockClkoutSelFlashClk* Flash clock.  
*kClockClkoutSelLpoClk* LPO clock.  
*kClockClkoutSelMcgIrClk* MCG out clock.  
*kClockClkoutSelRtc32kClk* RTC 32k clock.  
*kClockClkoutSelOsc0erClk* OSCERCLK0 clock.

### 53.2.55.3.10 enum clock\_rtcout\_src\_kw24d5\_t

Enumerator

*kClockRtcoutSrc1Hz* 1Hz clock  
*kClockRtcoutSrc32kHz* 32kHz clock

### 53.2.55.3.11 enum sim\_usbsstby\_mode\_kw24d5\_t

Enumerator

*kSimUsbsstbyNoRegulator* regulator not in standby during Stop modes

*kSimUsbsstbyWithRegulator* regulator in standby during Stop modes

### 53.2.55.3.12 enum sim\_usbvstby\_mode\_kw24d5\_t

Enumerator

*kSimUsbvstbyNoRegulator* regulator not in standby during VLPR and VLPW modes

*kSimUsbvstbyWithRegulator* regulator in standby during VLPR and VLPW modes

### 53.2.55.3.13 enum sim\_adc\_pretrg\_sel\_kw24d5\_t

Enumerator

*kSimAdcPretrgSelA* Pre-trigger A selected for ADCx.

*kSimAdcPretrgSelB* Pre-trigger B selected for ADCx.

### 53.2.55.3.14 enum sim\_adc\_trg\_sel\_kw24d5\_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.

### 53.2.55.3.15 enum sim\_uart\_rxsrc\_kw24d5\_t

Enumerator

*kSimUartRxsrcPin* UARTx\_RX Pin.

## SIM HAL driver

*kSimUartRxsrcCmp0* CMP0.

*kSimUartRxsrcCmp1* CMP1.

### 53.2.55.3.16 enum sim\_uart\_txsrc\_kw24d5\_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.

### 53.2.55.3.17 enum sim\_ftm\_trg\_src\_kw24d5\_t

Enumerator

*kSimFtmTrgSrc0* FlexTimer x trigger y select 0.

*kSimFtmTrgSrc1* FlexTimer x trigger y select 1.

### 53.2.55.3.18 enum sim\_ftm\_clk\_sel\_kw24d5\_t

Enumerator

*kSimFtmClkSel0* FTM CLKIN0 pin.

*kSimFtmClkSel1* FTM CLKIN1 pin.

### 53.2.55.3.19 enum sim\_ftm\_ch\_src\_kw24d5\_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.

### 53.2.55.3.20 enum sim\_ftmflt\_sel\_kw24d5\_t

Enumerator

*kSimFtmFltSel0* FlexTimer x fault y select 0.

*kSimFtmFltSel1* FlexTimer x fault y select 1.



**53.2.55.3.21 enum sim\_tpm\_clk\_sel\_kw24d5\_t**

Enumerator

*kSimTpmClkSel0* Timer/PWM TPM\_CLKIN0 pin.*kSimTpmClkSel1* Timer/PWM TPM\_CLKIN1 pin.**53.2.55.3.22 enum sim\_tpm\_ch\_src\_kw24d5\_t**

Enumerator

*kSimTpmChSrc0* TPMx\_CH0 signal.*kSimTpmChSrc1* CMP0 output.**53.2.55.3.23 enum sim\_cmtuartpad\_strenght\_kw24d5\_t**

Enumerator

*kSimCmtuartSinglePad* Single-pad drive strength for CMT IRO or UART0\_TXD.*kSimCmtuartDualPad* Dual-pad drive strength for CMT IRO or UART0\_TXD.**53.2.55.3.24 enum sim\_ptd7pad\_strenght\_kw24d5\_t**

Enumerator

*kSimPtd7padSinglePad* Single-pad drive strength for PTD7.*kSimPtd7padDualPad* Dual-pad drive strength for PTD7.**53.2.55.3.25 enum sim\_clock\_gate\_name\_kw24d5\_t**



---

## Chapter 54 System Mode Controller (SMC)

### 54.1 Overview

The Kinetis SDK provides both HAL and Peripheral drivers for the System Mode Controller (SMC) block of Kinetis devices.

#### Modules

- [SMC HAL driver](#)

## SMC HAL driver

### 54.2 SMC HAL driver

#### 54.2.1 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.

#### 54.2.2 Power Mode Configuration APIs

The function [SMC\\_HAL\\_SetMode\(\)](#) configures the power mode base on configuration. If could not switch to the target mode directly, this function could check internally and choose the right path.

If an interrupt or a reset occurs during a stop entry sequence, the SMC can abort the transition early without completely entering the stop mode. The function [SMC\\_HAL\\_IsStopAbort\(\)](#) checks whether the previous stop mode entry was aborted.

To get current power mode, please use the function [SMC\\_HAL\\_GetStat\(\)](#).

This is an example of the SMC manager APIs.

```
#include "fsl_smc_hal.h"

// power mode config structure //
smc_power_mode_config_t smcConfig;

// power mode and option mode //
smcConfig.powerModeName = kPowerModeRun;

// set the power mode //
SMC_HAL_SetMode(&smcConfig);
```

#### 54.2.3 Power Mode Protection APIs

To configure the allowed power mode, please use the function [SMC\\_HAL\\_SetProtection\(\)](#) and pass the allowed power modes. This function should only be called once after system reset.

There is an example shows how to use this function:

```
#include "fsl_smc_hal.h"

// Allow LLS mode and VLLS mode.
SMC_HAL_SetProtection(SMC, kAllowPowerModeLls |
 kAllowPowerModeVlls);
```

At the same time, the function [SMC\\_HAL\\_GetProtection\(\)](#) could check whether the specified modes are allowed.

## Files

- file `fsl_smc_hal.h`

## Data Structures

- 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` = 0x01U,  
`kStatStop` = 0x02U,  
`kStatVlpr` = 0x04U,  
`kStatVlpw` = 0x08U,  
`kStatVlps` = 0x10U,  
`kStatVlls` = 0x40U }  
*Power Modes in PMSTAT.*
- enum `power_modes_protect_t` {  
`kAllowPowerModeVlls` = `SMC_PMPROT_AVLLS_MASK`,  
`kAllowPowerModeVlpr` = `SMC_PMPROT_AVLP_MASK`,  
`kAllowPowerModeAll` }  
*Power Modes Protection.*
- enum `smc_run_mode_t` {  
`kSmcRun` ,  
`kSmcVlpr` }  
*Run mode definition.*
- enum `smc_stop_mode_t` {  
`kSmcStop` = 0U,  
`kSmcReservedStop1` = 1U,  
`kSmcVlps` = 2U,  
`kSmcVlls` = 4U }  
*Stop mode definition.*
- enum `smc_stop_submode_t` {

## SMC HAL driver

```
kSmcStopSub0,
kSmcStopSub1,
kSmcStopSub2,
kSmcStopSub3 }
```

*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.*

## System mode controller APIs

- `smc_hal_error_code_t SMC_HAL_SetMode` (`SMC_Type *base`, `const smc_power_mode_config_t *powerModeConfig`)  
*Configures the power mode.*
- static void `SMC_HAL_SetProtection` (`SMC_Type *base`, `uint8_t allowedModes`)  
*Configures all power mode protection settings.*
- static `uint8_t SMC_HAL_GetProtection` (`SMC_Type *base`, `uint8_t modes`)  
*Get the power mode protection setting.*
- static `bool SMC_HAL_IsStopAbort` (`SMC_Type *base`)  
*Check whether previous stop mode entry was successful.*
- `power_mode_stat_t SMC_HAL_GetStat` (`SMC_Type *base`)  
*Gets the current power mode status.*

## 54.2.4 Data Structure Documentation

### 54.2.4.1 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.*

## 54.2.5 Enumeration Type Documentation

### 54.2.5.1 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.

### 54.2.5.2 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

### 54.2.5.3 enum power\_modes\_protect\_t

Enumerator

- kAllowPowerModeVlls* Allow Very-Low-Leakage Stop Mode.
- kAllowPowerModeVlp* Allow Very-Low-Power Modes.
- kAllowPowerModeAll* Allow all power modes.

## SMC HAL driver

### 54.2.5.4 enum smc\_run\_mode\_t

Enumerator

- kSmcRun* normal RUN mode
- kSmcVlpr* Very-Low-Power RUN mode.

### 54.2.5.5 enum smc\_stop\_mode\_t

Enumerator

- kSmcStop* Normal STOP mode.
- kSmcReservedStop1* Reserved.
- kSmcVlps* Very-Low-Power STOP mode.
- kSmcVlls* Very-Low-Leakage Stop mode.

### 54.2.5.6 enum smc\_stop\_submode\_t

Enumerator

- kSmcStopSub0* Stop submode 0, for VLLS0/LLS0.
- kSmcStopSub1* Stop submode 1, for VLLS1/LLS1.
- kSmcStopSub2* Stop submode 2, for VLLS2/LLS2.
- kSmcStopSub3* Stop submode 3, for VLLS3/LLS3.

### 54.2.5.7 enum smc\_lpwui\_option\_t

Enumerator

- kSmcLpwuiEnabled* Low Power Wake Up on Interrupt enabled.
- kSmcLpwuiDisabled* Low Power Wake Up on Interrupt disabled.

### 54.2.5.8 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.



### 54.2.5.9 enum smc\_por\_option\_t

Enumerator

- kSmcPorEnabled* POR detect circuit is enabled in VLLS0.
- kSmcPorDisabled* POR detect circuit is disabled in VLLS0.

### 54.2.5.10 enum smc\_ram2\_option\_t

Enumerator

- kSmcRam2DisPowered* RAM2 not powered in LLS2/VLLS2.
- kSmcRam2Powered* RAM2 powered in LLS2/VLLS2.

### 54.2.5.11 enum smc\_lpo\_option\_t

Enumerator

- kSmcLpoEnabled* LPO clock is enabled in LLS/VLLSx.
- kSmcLpoDisabled* LPO clock is disabled in LLS/VLLSx.

## 54.2.6 Function Documentation

### 54.2.6.1 smc\_hal\_error\_code\_t SMC\_HAL\_SetMode ( SMC\_Type \* base, const smc\_power\_mode\_config\_t \* powerModeConfig )

This function configures the power mode base on configuration structure, if could not switch to the target mode directly, this function could check internally and choose the right path.

Parameters

|                         |                                                                            |
|-------------------------|----------------------------------------------------------------------------|
| <i>base</i>             | Base address for current SMC instance.                                     |
| <i>powerMode-Config</i> | Power mode configuration structure <a href="#">smc_power_mode_config_t</a> |

Returns

SMC error code.

## SMC HAL driver

**54.2.6.2 static void SMC\_HAL\_SetProtection ( SMC\_Type \* *base*, uint8\_t *allowedModes* ) [inline], [static]**

This function configures the power mode protection settings for supported power modes in the specified chip family. The available power modes are defined in the `power_modes_protect_t`. This should be done at an early system level initialization stage. See the reference manual for details. This register can only write once after the power reset.

The allowed modes are passed as bit map, for example, to allow LLS and VLLS, please use `SMC_HAL_SetProtection(SMC, kAllowPowerModeLls | kAllowPowerModeVlls)`. To allow all modes, please use `SMC_HAL_SetProtection(SMC, kAllowPowerModeAll)`.

Parameters

|                     |                                        |
|---------------------|----------------------------------------|
| <i>base</i>         | Base address for current SMC instance. |
| <i>allowedModes</i> | Bitmap of the allowed power modes.     |

**54.2.6.3 static uint8\_t SMC\_HAL\_GetProtection ( SMC\_Type \* *base*, uint8\_t *modes* ) [inline], [static]**

This function checks whether the power modes are allowed. The modes to check are passed as bit map, for example, to check LLS and VLLS, please use `SMC_HAL_GetProtection(SMC, kAllowPowerModeLls | kAllowPowerModeVlls)`. To test all modes, please use `SMC_HAL_GetProtection(SMC, kAllowPowerModeAll)`.

Parameters

|              |                                        |
|--------------|----------------------------------------|
| <i>base</i>  | Base address for current SMC instance. |
| <i>modes</i> | Bitmap of the power modes to check.    |

Returns

Bitmap of the allowed power modes.

**54.2.6.4 static bool SMC\_HAL\_IsStopAbort ( SMC\_Type \* *base* ) [inline], [static]**

Parameters

---

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | Base address for current SMC instance. |
|-------------|----------------------------------------|

Return values

|              |                                              |
|--------------|----------------------------------------------|
| <i>true</i>  | The previous stop mode entry was aborted.    |
| <i>false</i> | The previous stop mode entry was successful. |

#### 54.2.6.5 `power_mode_stat_t SMC_HAL_GetStat ( SMC_Type * base )`

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>base</i> | Base address for current SMC instance. |
|-------------|----------------------------------------|

Returns

Current power mode status.



## Chapter 55

# 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.

### 55.1 Overview

#### Modules

- [K02F12810](#)  
*The data structure definition for K02F12810 clock manager.*
- [K10D10](#)  
*The data structure definition for K10D10 clock manager.*
- [K11DA5](#)  
*The data structure definition for K11DA5 clock manager.*
- [K128T7](#)  
*The data structure definition for K128T7 clock manager.*
- [K20D10](#)  
*The data structure definition for K20D10 clock manager.*
- [K21DA5](#)  
*The data structure definition for K21DA5 clock manager.*
- [K21FA12](#)  
*The data structure definition for K21FA12 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.*
- [K30D10](#)  
*The data structure definition for K30D10 clock manager.*
- [K40D10](#)  
*The data structure definition for K40D10 clock manager.*
- [K50D10](#)  
*The data structure definition for K50D10 clock manager.*
- [K51D10](#)  
*The data structure definition for K51D10 clock manager.*
- [K52D10](#)  
*The data structure definition for K52D10 clock manager.*
- [K53D10](#)  
*The data structure definition for K53D10 clock manager.*
- [K60D10](#)

## Overview

- [K63F12](#)  
*The data structure definition for K60D10 clock manager.*
- [K64F12](#)  
*The data structure definition for K63F12 clock manager.*
- [KL02Z4](#)  
*The data structure definition for K64F12 clock manager.*
- [KL03Z4](#)  
*The data structure definition for KL02Z4 clock manager.*
- [KL127Z644](#)  
*The data structure definition for KL03Z4 clock manager.*
- [KL13Z644](#)  
*The data structure definition for KL127Z644 clock manager.*
- [KL14Z4](#)  
*The data structure definition for KL13Z644 clock manager.*
- [KL15Z4](#)  
*The data structure definition for KL14Z4 clock manager.*
- [KL16Z4](#)  
*The data structure definition for KL15Z4 clock manager.*
- [KL17Z4](#)  
*The data structure definition for KL16Z4 clock manager.*
- [KL17Z644](#)  
*The data structure definition for KL17Z4 clock manager.*
- [KL24Z4](#)  
*The data structure definition for KL17Z644 clock manager.*
- [KL25Z4](#)  
*The data structure definition for KL24Z4 clock manager.*
- [KL26Z4](#)  
*The data structure definition for KL25Z4 clock manager.*
- [KL27Z4](#)  
*The data structure definition for KL26Z4 clock manager.*
- [KL33Z4](#)  
*The data structure definition for KL27Z4 clock manager.*
- [KL33Z644](#)  
*The data structure definition for KL33Z4 clock manager.*
- [KL34Z4](#)  
*The data structure definition for KL33Z644 clock manager.*
- [KL36Z4](#)  
*The data structure definition for KL34Z4 clock manager.*
- [KL43Z4](#)  
*The data structure definition for KL36Z4 clock manager.*
- [KL46Z4](#)  
*The data structure definition for KL43Z4 clock manager.*
- [KV10Z7](#)  
*The data structure definition for KL46Z4 clock manager.*
- [KV30F12810](#)  
*The data structure definition for KV10Z7 clock manager.*
- [KV31F12810](#)  
*The data structure definition for KV30F12810 clock manager.*
- [KV31F25612](#)  
*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.
- [KW21D5](#)  
The data structure definition for KW21D5 clock manager.
- [KW22D5](#)  
The data structure definition for KW22D5 clock manager.
- [KW24D5](#)  
The data structure definition for KW24D5 clock manager.

## Files

- file [fsl\\_clock\\_manager.h](#)
- file [fsl\\_clock\\_MKV40F15.h](#)
- file [fsl\\_clock\\_MKV43F15.h](#)
- file [fsl\\_clock\\_MKV44F15.h](#)
- file [fsl\\_clock\\_MKV45F15.h](#)
- file [fsl\\_clock\\_MKV46F15.h](#)

## Data Structures

- struct [oscer\\_config\\_t](#)  
OSC configuration for OSCERCLK. [More...](#)
- struct [osc\\_user\\_config\\_t](#)  
OSC Initialization Configuration Structure. [More...](#)
- struct [rtc\\_osc\\_user\\_config\\_t](#)  
RTC OSC Initialization Configuration Structure. [More...](#)
- 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...](#)
- struct [sim\\_config\\_t](#)  
SIM configuration structure for dynamic clock setting. [More...](#)
- struct [clock\\_name\\_config\\_t](#)  
Clock name configuration table 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)

## Overview

*Type of clock callback functions.*

## Enumerations

- enum `clock_systick_src_t` {  
`kClockSystickSrcExtRef = 0U`,  
`kClockSystickSrcCore = 1U` }  
*Systick clock source selection.*
- enum `clock_names_t` {  
`kCoreClock`,  
`kSystemClock`,  
`kPlatformClock`,  
`kBusClock`,  
`kFlexBusClock`,  
`kFlashClock`,  
`kFastPeripheralClock`,  
`kSystickClock`,  
`kOsc32kClock`,  
`kOsc0ErClock`,  
`kOsc1ErClock`,  
`kOsc0ErClockUndiv`,  
`kIrc48mClock`,  
`kRtcoutClock`,  
`kMcgFfClock`,  
`kMcgFllClock`,  
`kMcgPll0Clock`,  
`kMcgPll1Clock`,  
`kMcgExtPllClock`,  
`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` }





## Overview

- static void [CLOCK\\_SYS\\_SetPllflSel](#) (clock\_pllfl\_sel\_t setting)  
*Set PLL/FLL clock selection.*
- static [clock\\_pllfl\\_sel\\_t CLOCK\\_SYS\\_GetPllflSel](#) (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.*

  - static uint32\_t [CLOCK\\_SYS\\_GetFtmSystemClockFreq](#) (uint32\_t instance)
- Gets FTM's system 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)
- 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.*

## Overview

- 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\\_EnableLptmrClock](#) (uint32\_t instance)  
*Enable the clock for LPTIMER module.*
- static void [CLOCK\\_SYS\\_DisableLptmrClock](#) (uint32\_t instance)  
*Disable the clock for LPTIMER module.*
- static bool [CLOCK\\_SYS\\_GetLptmrGateCmd](#) (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.*
- static void [CLOCK\\_SYS\\_SetFtmExternalFreq](#) (uint32\_t srcInstance, uint32\_t freq)

*Set the FTM external clock frequency(FTM\_CLKx).*
- 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 uint32\_t [CLOCK\\_SYS\\_GetRtcFreq](#) (uint32\_t instance)

*Gets RTC input clock frequency.*
- 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 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.*
- uint32\_t [CLOCK\\_SYS\\_GetFlexcanFreq](#) (uint32\_t instance, clock\_flexcan\_src\_t flexcanSrc)

*Gets FLEXCAN 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)

*Gets ethernet timestamp clock frequency.*
- static uint32\_t [CLOCK\\_SYS\\_GetCmtFreq](#) (uint32\_t instance)

*Gets the clock frequency for CMT module.*
- 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.*

## Overview

- static `clock_sdhc_src_t` `CLOCK_SYS_GetSdhcSrc` (`uint32_t` instance)  
*Get the SDHC clock source selection.*
- `uint32_t` `CLOCK_SYS_GetSaiFreq` (`uint32_t` instance, `clock_sai_src_t` `saiSrc`)  
*Gets the clock frequency for SAI.*
- 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_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.*
- 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_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_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_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_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.*
- void `CLOCK_SYS_EnableFlexcanClock` (`uint32_t` instance)  
*Enable the clock for FLEXCAN module.*
- void `CLOCK_SYS_DisableFlexcanClock` (`uint32_t` instance)

- Disable the clock for FLEXCAN module.*

  - 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_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).*
  - static void `CLOCK_SYS_SetSdhcExternalFreq` (uint32\_t srcInstance, uint32\_t freq)  
*Set the SDHC external clock frequency(SDHC\_CLKIN).*
  - 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 usbdiv, uint8\_t usbfrac)  
*Set USB FS divider setting.*
  - static void `CLOCK_SYS_GetUsbfsDiv` (uint32\_t instance, uint8\_t \*usbdiv, uint8\_t \*usbfrac)  
*Get USB FS divider setting.*
  - 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_EnableUsbdcdClock` (uint32\_t instance)  
*Enable the clock for USBDCD module.*
  - static void `CLOCK_SYS_DisableUsbdcdClock` (uint32\_t instance)  
*Disable the clock for USBDCD module.*
  - static bool `CLOCK_SYS_GetUsbdcdGateCmd` (uint32\_t instance)  
*Get the the clock gate state for USBDCD 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.*
  - void `CLOCK_SYS_Osc0Deinit` (void)  
*Deinitialize OSC0.*
  - static void `CLOCK_SYS_SetUsbExternalFreq` (uint32\_t srcInstance, uint32\_t freq)  
*Set the USB external clock frequency(USB\_CLKIN).*

## Overview

- 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)  
*Sets the clock source for LPUART module.*
- `uint32_t` `CLOCK_SYS_GetLpuartFreq` (`uint32_t` instance)  
*Gets the clock frequency for LPUART 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.*
- `uint32_t` `CLOCK_SYS_GetPIIFLIDivClockFreq` (void)  
*Gets the PLL/FLL clock divided by the fractional divider.*
- `uint32_t` `CLOCK_SYS_GetTpmFreq` (`uint32_t` instance)  
*Gets TPM clock frequency.*
- 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.*
- static `clock_usbhs_slowclk_src_t` `CLOCK_SYS_GetUsbhsSlowClockSrc` (`uint32_t` instance)  
*Gets the slow clock source for USB HS/USB PHY module.*
- static void `CLOCK_SYS_SetUsbhsSlowClockSrc` (`uint32_t` instance, `clock_usbhs_slowclk_src_t` usbhsSrc)  
*Sets the clock source for USB HS/USB PHY module.*
- `uint32_t` `CLOCK_SYS_GetUsbhsSlowClockFreq` (`uint32_t` instance)  
*Gets the slow clock frequency for USB HS/USB PHY module.*
- static void `CLOCK_SYS_EnableSdramcClock` (`uint32_t` instance)  
*Enable the clock for SDRAMC module.*
- static void `CLOCK_SYS_DisableSdramcClock` (`uint32_t` instance)  
*Disable the clock for SDRAMC module.*
- static bool `CLOCK_SYS_GetSdramcGateCmd` (`uint32_t` instance)  
*Get the the clock gate state for SDRAMC module.*
- 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_EnableUsbhsClock` (`uint32_t` instance)  
*Enable the clock for USBHS module.*
- static void `CLOCK_SYS_DisableUsbhsClock` (`uint32_t` instance)  
*Disable the clock for USBHS module.*
- static bool `CLOCK_SYS_GetUsbhsGateCmd` (`uint32_t` instance)  
*Get the the clock gate state for USBHS module.*



- static void [CLOCK\\_SYS\\_EnableUsbphyClock](#) (uint32\_t instance)  
*Enable the clock for USBPHY module.*
- static void [CLOCK\\_SYS\\_DisableUsbphyClock](#) (uint32\_t instance)  
*Disable the clock for USBPHY module.*
- static bool [CLOCK\\_SYS\\_GetUsbphyGateCmd](#) (uint32\_t instance)  
*Get the the clock gate state for USBPHY module.*
- static void [CLOCK\\_SYS\\_EnableUsbhsdcdClock](#) (uint32\_t instance)  
*Enable the clock for USBHSDCD module.*
- static void [CLOCK\\_SYS\\_DisableUsbhsdcdClock](#) (uint32\_t instance)  
*Disable the clock for USBHSDCD module.*
- static bool [CLOCK\\_SYS\\_GetUsbhsdcdGateCmd](#) (uint32\_t instance)  
*Get the the clock gate state for USBHSDCD module.*
- uint32\_t [CLOCK\\_SYS\\_GetFllClockFreq](#) (void)  
*Get the MCGFLLCLK clock frequency.*
- uint32\_t [CLOCK\\_SYS\\_GetCopFreq](#) (uint32\_t instance, clock\_cop\_src\_t copSrc)  
*Gets the COP clock frequency.*
- uint32\_t [CLOCK\\_SYS\\_GetLpsciFreq](#) (uint32\_t instance)  
*Gets the clock frequency for LPSCI module.*
- static void [CLOCK\\_SYS\\_SetLpsciSrc](#) (uint32\_t instance, clock\_lpsci\_src\_t lpsciSrc)  
*Set the LPSCI clock source selection.*
- static clock\_lpsci\_src\_t [CLOCK\\_SYS\\_GetLpsciSrc](#) (uint32\_t instance)  
*Get the LPSCI clock source selection.*
- void [CLOCK\\_SYS\\_EnableLpsciClock](#) (uint32\_t instance)  
*Enable the clock for LPSCI module.*
- void [CLOCK\\_SYS\\_DisableLpsciClock](#) (uint32\_t instance)  
*Disable the clock for LPSCI module.*
- bool [CLOCK\\_SYS\\_GetLpsciGateCmd](#) (uint32\_t instance)  
*Get the the clock gate state for LPSCI 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](#) (void)  
*Gets COP clock frequency.*
- static void [CLOCK\\_SYS\\_SetCopSrc](#) (clock\_cop\_src\_t copSrc)  
*Set the COP clock source selection.*
- static clock\_cop\_src\_t [CLOCK\\_SYS\\_GetCopSrc](#) (void)  
*Get the COP clock source selection.*
- uint32\_t [CLOCK\\_SYS\\_GetFlexioFreq](#) (uint32\_t instance)  
*Gets FLEXIO clock frequency.*
- static void [CLOCK\\_SYS\\_SetFlexioSrc](#) (uint32\_t instance, clock\_flexio\_src\_t flexioSrc)  
*Set the FLEXIO clock source selection.*
- static clock\_flexio\_src\_t [CLOCK\\_SYS\\_GetFlexioSrc](#) (uint32\_t instance)  
*Get the FLEXIO clock source selection.*
- static void [CLOCK\\_SYS\\_EnableFlexioClock](#) (uint32\_t instance)  
*Enable the clock for FLEXIO module.*
- static void [CLOCK\\_SYS\\_DisableFlexioClock](#) (uint32\_t instance)  
*Disable the clock for FLEXIO module.*
- static bool [CLOCK\\_SYS\\_GetFlexioGateCmd](#) (uint32\_t instance)  
*Get the clock gate state for FLEXIO module.*
- void [CLOCK\\_SYS\\_EnableXrdcClock](#) (uint32\_t instance)  
*Enable the clock for XRDC module.*
- void [CLOCK\\_SYS\\_DisableXrdcClock](#) (uint32\_t instance)

## Overview

- Disable the clock for XRDC module.*

  - bool [CLOCK\\_SYS\\_GetXrdcGateCmd](#) (uint32\_t instance)

*Get the the clock gate state for XRDC module.*
- void [CLOCK\\_SYS\\_EnableSemaClock](#) (uint32\_t instance)

*Enable the clock for SEMA module.*
- void [CLOCK\\_SYS\\_DisableSemaClock](#) (uint32\_t instance)

*Disable the clock for SEMA module.*
- bool [CLOCK\\_SYS\\_GetSemaGateCmd](#) (uint32\_t instance)

*Get the the clock gate state for SEMA module.*
- void [CLOCK\\_SYS\\_EnableFlashClock](#) (uint32\_t instance)

*Enable the clock for FLASH module.*
- void [CLOCK\\_SYS\\_DisableFlashClock](#) (uint32\_t instance)

*Disable the clock for FLASH module.*
- bool [CLOCK\\_SYS\\_GetFlashGateCmd](#) (uint32\_t instance)

*Get the the clock gate state for FLASH module.*
- void [CLOCK\\_SYS\\_EnableMuClock](#) (uint32\_t instance)

*Enable the clock for MU module.*
- void [CLOCK\\_SYS\\_DisableMuClock](#) (uint32\_t instance)

*Disable the clock for MU module.*
- bool [CLOCK\\_SYS\\_GetMuGateCmd](#) (uint32\_t instance)

*Get the the clock gate state for MU module.*
- void [CLOCK\\_SYS\\_EnableIntmuxClock](#) (uint32\_t instance)

*Enable the clock for INTMUX module.*
- void [CLOCK\\_SYS\\_DisableIntmuxClock](#) (uint32\_t instance)

*Disable the clock for INTMUX module.*
- bool [CLOCK\\_SYS\\_GetIntmuxGateCmd](#) (uint32\_t instance)

*Get the the clock gate state for INTMUX module.*
- clock\_pit\_src\_t [CLOCK\\_SYS\\_GetPitSrc](#) (uint32\_t instance)

*Gets the clock source for PIT module.*
- void [CLOCK\\_SYS\\_SetPitSrc](#) (uint32\_t instance, clock\_pit\_src\_t pitSrc)

*Sets the clock source for PIT module.*
- void [CLOCK\\_SYS\\_EnableLpspiClock](#) (uint32\_t instance)

*Enable the clock for LPSPI module.*
- void [CLOCK\\_SYS\\_DisableLpspiClock](#) (uint32\_t instance)

*Disable the clock for LPSPI module.*
- bool [CLOCK\\_SYS\\_GetLpspiGateCmd](#) (uint32\_t instance)

*Get the the clock gate state for LPSPI module.*
- clock\_lpspi\_src\_t [CLOCK\\_SYS\\_GetLpspiSrc](#) (uint32\_t instance)

*Gets the clock source for LPSPI module.*
- void [CLOCK\\_SYS\\_SetLpspiSrc](#) (uint32\_t instance, clock\_lpspi\_src\_t lpspiSrc)

*Sets the clock source for LPSPI module.*
- uint32\_t [CLOCK\\_SYS\\_GetLpspiFreq](#) (uint32\_t instance)

*Gets the clock frequency for LPSPI module.*
- void [CLOCK\\_SYS\\_EnableLpi2cClock](#) (uint32\_t instance)

*Enable the clock for LPI2C module.*
- void [CLOCK\\_SYS\\_DisableLpi2cClock](#) (uint32\_t instance)

*Disable the clock for LPI2C module.*
- bool [CLOCK\\_SYS\\_GetLpi2cGateCmd](#) (uint32\_t instance)

*Get the the clock gate state for LPI2C module.*
- clock\_lpi2c\_src\_t [CLOCK\\_SYS\\_GetLpi2cSrc](#) (uint32\_t instance)

*Gets the clock source for LPI2C module.*

- void [CLOCK\\_SYS\\_SetLpi2cSrc](#) (uint32\_t instance, clock\_lpi2c\_src\_t lpi2cSrc)  
*Sets the clock source for LPI2C module.*
- uint32\_t [CLOCK\\_SYS\\_GetLpi2cFreq](#) (uint32\_t instance)  
*Gets the clock frequency for LPI2C module.*
- void [CLOCK\\_SYS\\_EnableEvmsimClock](#) (uint32\_t instance)  
*Enable the clock for EVMSIM module.*
- void [CLOCK\\_SYS\\_DisableEvmsimClock](#) (uint32\_t instance)  
*Disable the clock for EVMSIM module.*
- bool [CLOCK\\_SYS\\_GetEvmsimGateCmd](#) (uint32\_t instance)  
*Get the the clock gate state for EVMSIM module.*
- clock\_evmsim\_src\_t [CLOCK\\_SYS\\_GetEvmsimSrc](#) (uint32\_t instance)  
*Gets the clock source for EVMSIM module.*
- void [CLOCK\\_SYS\\_SetEvmsimSrc](#) (uint32\_t instance, clock\_evmsim\_src\_t evmsimSrc)  
*Sets the clock source for EVMSIM module.*
- uint32\_t [CLOCK\\_SYS\\_GetEvmsimFreq](#) (uint32\_t instance)  
*Gets the clock frequency for EVMSIM module.*
- void [CLOCK\\_SYS\\_EnableAtxClock](#) (uint32\_t instance)  
*Enable the clock for ATX module.*
- void [CLOCK\\_SYS\\_DisableAtxClock](#) (uint32\_t instance)  
*Disable the clock for ATX module.*
- bool [CLOCK\\_SYS\\_GetAtxGateCmd](#) (uint32\_t instance)  
*Get the the clock gate state for ATX module.*
- void [CLOCK\\_SYS\\_EnableTrngClock](#) (uint32\_t instance)  
*Enable the clock for TRNG module.*
- void [CLOCK\\_SYS\\_DisableTrngClock](#) (uint32\_t instance)  
*Disable the clock for TRNG module.*
- bool [CLOCK\\_SYS\\_GetTrngGateCmd](#) (uint32\_t instance)  
*Get the the clock gate state for TRNG 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.*
- uint32\_t [CLOCK\\_SYS\\_GetFastPeripheralClockFreq](#) (void)  
*Gets the fast peripheral clock frequency.*
- 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\\_GetAdcFreq](#) (uint32\_t instance)  
*Gets the clock frequency for ADC module.*
- uint32\_t [CLOCK\\_SYS\\_GetPwmFreq](#) (uint32\_t instance)  
*Gets the clock frequency for eFlexPWM module.*
- uint32\_t [CLOCK\\_SYS\\_GetEncFreq](#) (uint32\_t instance)  
*Gets the clock frequency for ENC module.*
- uint32\_t [CLOCK\\_SYS\\_GetXbarFreq](#) (uint32\_t instance)  
*Gets the clock frequency for XBAR module.*
- uint32\_t [CLOCK\\_SYS\\_GetAoiFreq](#) (uint32\_t instance)  
*Gets the clock frequency for AOI module.*
- uint32\_t [CLOCK\\_SYS\\_GetFlexcanFreq](#) (uint8\_t instance, [clock\\_flexcan\\_src\\_t](#) flexcanSrc)

## Overview

- Gets FLEXCAN clock frequency.*

  - static void [CLOCK\\_SYS\\_EnablePwmClock](#) (uint32\_t instance)  
*Enable the clock for eFlexPWM module.*
  - static void [CLOCK\\_SYS\\_DisablePwmClock](#) (uint32\_t instance)  
*Disable the clock for eFlexPWM module.*
  - static bool [CLOCK\\_SYS\\_GetPwmGateCmd](#) (uint32\_t instance)  
*Get the the clock gate state for eFlexPWM module.*
  - static void [CLOCK\\_SYS\\_EnableAoiClock](#) (uint32\_t instance)  
*Enable the clock for AOI module.*
  - static void [CLOCK\\_SYS\\_DisableAoiClock](#) (uint32\_t instance)  
*Disable the clock for AOI module.*
  - static bool [CLOCK\\_SYS\\_GetAoiGateCmd](#) (uint32\_t instance)  
*Get the the clock gate state for AOI module.*
  - static void [CLOCK\\_SYS\\_EnableXbarClock](#) (uint32\_t instance)  
*Enable the clock for XBAR module.*
  - static void [CLOCK\\_SYS\\_DisableXbarClock](#) (uint32\_t instance)  
*Disable the clock for XBAR module.*
  - static bool [CLOCK\\_SYS\\_GetXbarGateCmd](#) (uint32\_t instance)  
*Get the the clock gate state for XBAR module.*
  - static void [CLOCK\\_SYS\\_EnableEncClock](#) (uint32\_t instance)  
*Enable the clock for ENC module.*
  - static void [CLOCK\\_SYS\\_DisableEncClock](#) (uint32\_t instance)  
*Disable the clock for ENC module.*
  - static bool [CLOCK\\_SYS\\_GetEncGateCmd](#) (uint32\_t instance)  
*Get the the clock gate state for ENC module.*

## Variables

- SIM\_Type \*const [g\\_simBase](#) []  
*The register base of SIM module.*
- MCG\_Type \*const [g\\_mcgBase](#) []  
*The register base of MCG/MCG\_LITE module.*
- OSC\_Type \*const [g\\_oscBase](#) []  
*The register base of OSC module.*

## Dynamic clock setting

- [clock\\_manager\\_error\\_code\\_t](#) [CLOCK\\_SYS\\_Init](#) ([clock\\_manager\\_user\\_config\\_t](#) const \*\*clock-ConfigsPtr, 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(*fl-StableDelay)(void))`  
*Set MCG to some target mode.*

## OSC configuration

- `clock_manager_error_code_t CLOCK_SYS_OscInit (uint32_t instance, osc_user_config_t *config)`  
*Initialize OSC.*
- `void CLOCK_SYS_OscDeinit (uint32_t instance)`  
*Deinitialize OSC.*
- `void CLOCK_SYS_SetOscerConfiguration (uint32_t instance, oscer_config_t const *config)`  
*Configure the OSCERCLK.*

### 55.1.1 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);
```

## Overview

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, MCGPLLFLLCLK 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.

## 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 callback 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,
```

## Overview

```

 .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->notifyType)
 {
 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;
 break;
 }
 return ret;
}

```



## 55.2 Data Structure Documentation

### 55.2.1 struct oscr\_config\_t

#### Data Fields

- bool [enable](#)  
*OSKERCLK enable or not.*
- bool [enableInStop](#)  
*OSKERCLK enable or not in stop mode.*

#### 55.2.1.0.0.1 Field Documentation

##### 55.2.1.0.0.1.1 bool oscr\_config\_t::enable

##### 55.2.1.0.0.1.2 bool oscr\_config\_t::enableInStop

### 55.2.2 struct osc\_user\_config\_t

Defines the configuration data structure to initialize the OSC. When porting to a new board, please set the following members according to board setting:

1. freq: The external frequency.
2. hgo/range/erefs: These members should be set base on the board setting.

#### Data Fields

- uint32\_t [freq](#)  
*External clock frequency.*
- bool [enableCapacitor2p](#)  
*Enable 2pF capacitor load.*
- bool [enableCapacitor4p](#)  
*Enable 4pF capacitor load.*
- bool [enableCapacitor8p](#)  
*Enable 8pF capacitor load.*
- bool [enableCapacitor16p](#)  
*Enable 16pF capacitor load.*
- [osc\\_gain\\_t](#) [hgo](#)  
*High gain oscillator select.*
- [osc\\_range\\_t](#) [range](#)  
*Oscillator range setting.*
- [osc\\_src\\_t](#) [erefs](#)  
*External reference select.*
- [oscer\\_config\\_t](#) [oscerConfig](#)  
*Configuration for OSKERCLK.*

## Data Structure Documentation

### 55.2.2.0.0.2 Field Documentation

- 55.2.2.0.0.2.1 `uint32_t osc_user_config_t::freq`
- 55.2.2.0.0.2.2 `bool osc_user_config_t::enableCapacitor2p`
- 55.2.2.0.0.2.3 `bool osc_user_config_t::enableCapacitor4p`
- 55.2.2.0.0.2.4 `bool osc_user_config_t::enableCapacitor8p`
- 55.2.2.0.0.2.5 `bool osc_user_config_t::enableCapacitor16p`
- 55.2.2.0.0.2.6 `osc_gain_t osc_user_config_t::hgo`
- 55.2.2.0.0.2.7 `osc_range_t osc_user_config_t::range`
- 55.2.2.0.0.2.8 `osc_src_t osc_user_config_t::erefs`
- 55.2.2.0.0.2.9 `oscer_config_t osc_user_config_t::oscerConfig`

### 55.2.3 struct `rtc_osc_user_config_t`

Defines the configuration data structure to initialize the RTC OSC. When porting to a new board, please set the following members according to board setting:

1. `freq`: The external frequency for RTC.
2. `enableOSC`: RTC could use its dedicate OSC, or override the `OSC0` setting and use `OSC0`, or use external input clock directly. This is different by SOC and board setting, please set this correctly.

### Data Fields

- `uint32_t freq`  
*External clock frequency.*
- `bool enableCapacitor2p`  
*Enable 2pF capacitor load.*
- `bool enableCapacitor4p`  
*Enable 4pF capacitor load.*
- `bool enableCapacitor8p`  
*Enable 8pF capacitor load.*
- `bool enableCapacitor16p`  
*Enable 16pF capacitor load.*
- `bool enableOsc`  
*Enable OSC or use external clock directly.*
- `bool enableClockOutput`  
*Output clock to other peripherals or not.*

### 55.2.3.0.0.3 Field Documentation

55.2.3.0.0.3.1 `uint32_t rtc_osc_user_config_t::freq`

55.2.3.0.0.3.2 `bool rtc_osc_user_config_t::enableCapacitor2p`

55.2.3.0.0.3.3 `bool rtc_osc_user_config_t::enableCapacitor4p`

55.2.3.0.0.3.4 `bool rtc_osc_user_config_t::enableCapacitor8p`

55.2.3.0.0.3.5 `bool rtc_osc_user_config_t::enableCapacitor16p`

55.2.3.0.0.3.6 `bool rtc_osc_user_config_t::enableOsc`

55.2.3.0.0.3.7 `bool rtc_osc_user_config_t::enableClockOutput`

## 55.2.4 struct `mcg_config_t`

When porting to a new board, please set the following members according to board setting:

1. `frdiv`: If FLL uses the external reference clock, please set this value to make sure external reference clock divided by `frdiv` is in the range 31.25kHz to 39.0625kHz.
2. `prdiv0/vdiv0/prdiv1/vdiv1`: Please set these values for PLL, the PLL reference clock frequency after `prdiv` should be in the range of `FSL_FEATURE_MCG_PLL_REF_MIN` to `FSL_FEATURE_MCG_PLL_REF_MAX`.

## Data Fields

- `mcg_modes_t mcg_mode`  
*MCG mode.*
- `bool irclkEnable`  
*MCGIRCLK enable.*
- `bool irclkEnableInStop`  
*MCGIRCLK enable in stop mode.*
- `mcg_irc_mode_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].*

## Data Structure Documentation

### 55.2.4.0.0.4 Field Documentation

55.2.4.0.0.4.1 `mcg_modes_t mcg_config_t::mcg_mode`

55.2.4.0.0.4.2 `bool mcg_config_t::irclkEnable`

55.2.4.0.0.4.3 `bool mcg_config_t::irclkEnableInStop`

55.2.4.0.0.4.4 `mcg_irc_mode_t mcg_config_t::ircs`

55.2.4.0.0.4.5 `uint8_t mcg_config_t::fcrdiv`

55.2.4.0.0.4.6 `uint8_t mcg_config_t::frdiv`

55.2.4.0.0.4.7 `mcg_dco_range_select_t mcg_config_t::drs`

55.2.4.0.0.4.8 `mcg_dm32_select_t mcg_config_t::dm32`

### 55.2.5 struct `clock_manager_user_config_t`

#### Data Fields

- [mcg\\_config\\_t mcgConfig](#)  
*MCG configuration.*
- [oscer\\_config\\_t oscerConfig](#)  
*OSCEERCLK configuration.*
- [sim\\_config\\_t simConfig](#)  
*SIM configuration.*

### 55.2.5.0.0.5 Field Documentation

55.2.5.0.0.5.1 `mcg_config_t clock_manager_user_config_t::mcgConfig`

55.2.5.0.0.5.2 `oscer_config_t clock_manager_user_config_t::oscerConfig`

55.2.5.0.0.5.3 `sim_config_t clock_manager_user_config_t::simConfig`

### 55.2.6 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.*

### 55.2.6.0.0.6 Field Documentation

55.2.6.0.0.6.1 `uint8_t clock_notify_struct_t::targetClockConfigIndex`

55.2.6.0.0.6.2 `clock_manager_policy_t clock_notify_struct_t::policy`

55.2.6.0.0.6.3 `clock_manager_notify_t clock_notify_struct_t::notifyType`

## 55.2.7 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.*

### 55.2.7.0.0.7 Field Documentation

55.2.7.0.0.7.1 `clock_manager_callback_t clock_manager_callback_user_config_t::callback`

55.2.7.0.0.7.2 `clock_manager_callback_type_t clock_manager_callback_user_config_t::callbackType`

55.2.7.0.0.7.3 `void* clock_manager_callback_user_config_t::callbackData`

## 55.2.8 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.*

## Macro Definition Documentation

### 55.2.8.0.0.8 Field Documentation

55.2.8.0.0.8.1 `clock_manager_user_config_t const** clock_manager_state_t::configTable`

55.2.8.0.0.8.2 `uint8_t clock_manager_state_t::clockConfigNum`

55.2.8.0.0.8.3 `uint8_t clock_manager_state_t::curConfigIndex`

55.2.8.0.0.8.4 `clock_manager_callback_user_config_t** clock_manager_state_t::callbackConfig`

55.2.8.0.0.8.5 `uint8_t clock_manager_state_t::callbackNum`

55.2.8.0.0.8.6 `uint8_t clock_manager_state_t::errorCallbackIndex`

### 55.2.9 struct `sim_config_t`

#### Data Fields

- [clock\\_er32k\\_src\\_t er32kSrc](#)  
*ERCLK32K source selection.*
- `uint8_t outdiv4`  
*OUTDIV setting.*

### 55.2.9.0.0.9 Field Documentation

55.2.9.0.0.9.1 `clock_er32k_src_t sim_config_t::er32kSrc`

55.2.9.0.0.9.2 `uint8_t sim_config_t::outdiv4`

### 55.2.10 struct `clock_name_config_t`

#### Data Fields

- `bool useOtherRefClock`  
*if it uses the other ref clock*
- [clock\\_names\\_t otherRefClockName](#)  
*other ref clock name*
- [clock\\_divider\\_names\\_t dividerName](#)  
*clock divider name*

## 55.3 Macro Definition Documentation

55.3.1 `#define CPU_LPO_CLK_HZ 1000U`

## 55.4 Typedef Documentation

### 55.4.1 typedef clock\_manager\_error\_code\_t(\* clock\_manager\_callback\_t)(clock\_notify\_struct\_t \*notify, void \*callbackData)

## 55.5 Enumeration Type Documentation

### 55.5.1 enum clock\_systick\_src\_t

Enumerator

- kClockSystickSrcExtRef* Use external reference clock.
- kClockSystickSrcCore* Use processor clock (Core clock).

### 55.5.2 enum clock\_names\_t

Enumerator

- kCoreClock* Core clock.
- kSystemClock* System clock.
- kPlatformClock* Platform clock.
- kBusClock* Bus clock.
- kFlexBusClock* FlexBus clock.
- kFlashClock* Flash clock.
- kFastPeripheralClock* Flash peripheral clock.
- kSystickClock* Clock for systick.
- 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.
- kMcgExtPllClock* EXT\_PLLCLK.
- kMcgOutClock* MCGOUTCLK.
- kMcgIrClock* MCGIRCLK.
- kLpoClock* LPO clock.

## Function Documentation

### 55.5.3 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.

### 55.5.4 enum clock\_manager\_notify\_t

Enumerator

*kClockManagerNotifyRecover* Notify IP to recover to previous work state.  
*kClockManagerNotifyBefore* Notify IP that system will change clock setting.  
*kClockManagerNotifyAfter* Notify IP that have changed to new clock setting.

### 55.5.5 enum clock\_manager\_callback\_type\_t

Enumerator

*kClockManagerCallbackBefore* Callback handles BEFORE notification.  
*kClockManagerCallbackAfter* Callback handles AFTER notification.  
*kClockManagerCallbackBeforeAfter* Callback handles BEFORE and AFTER notification.

### 55.5.6 enum clock\_manager\_policy\_t

Enumerator

*kClockManagerPolicyAgreement* Clock transfers gracefully.  
*kClockManagerPolicyForcible* Clock transfers forcefully.

## 55.6 Function Documentation

### 55.6.1 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 <code>clock_names_t</code> |
| <i>frequency</i> | Returned clock frequency value in Hertz           |

Returns

status Error code defined in `clock_manager_error_code_t`

### 55.6.2 `uint32_t CLOCK_SYS_GetCoreClockFreq ( void )`

This function gets the core clock frequency.

Returns

Current core clock frequency.

### 55.6.3 `uint32_t CLOCK_SYS_GetSystemClockFreq ( void )`

This function gets the system clock frequency.

Returns

Current system clock frequency.

### 55.6.4 `uint32_t CLOCK_SYS_GetBusClockFreq ( void )`

This function gets the bus clock frequency.

Returns

Current bus clock frequency.

## Function Documentation

### 55.6.5 uint32\_t CLOCK\_SYS\_GetFlashClockFreq ( void )

This function gets the flash clock frequency.

Returns

Current flash clock frequency.

### 55.6.6 static uint32\_t CLOCK\_SYS\_GetLpoClockFreq ( void ) [inline], [static]

This function gets the LPO clock frequency.

Returns

Current clock frequency.

### 55.6.7 static void CLOCK\_SYS\_SetSystickSrc ( clock\_systick\_src\_t src ) [inline], [static]

This function selects the clock source for systick, systick clock source could be external reference clock or processor clock. Please check reference manual for details.

Parameters

|            |                           |
|------------|---------------------------|
| <i>src</i> | Clock source for systick. |
|------------|---------------------------|

### 55.6.8 static uint32\_t CLOCK\_SYS\_GetSystickFreq ( void ) [inline], [static]

This function gets the clock frequency for systick. Systick clock source could be external reference clock or processor clock. Please check reference manual for details.

Returns

Clock frequency for systick.

**55.6.9** `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.

## Function Documentation

### Parameters

|                        |                                              |
|------------------------|----------------------------------------------|
| <i>clockConfigsPtr</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>callbacksNumber</i> | Number of callback configurations in table.  |

### Returns

Error code.

### 55.6.10 `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>targetConfigIndex</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.

### 55.6.11 `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.

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.

**55.6.12 uint8\_t CLOCK\_SYS\_GetCurrentConfiguration ( void )**

Returns

Current clock configuration index.

**55.6.13 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.

**55.6.14 mcg\_mode\_error\_t CLOCK\_SYS\_SetMcgMode ( mcg\_config\_t const \* targetConfig, void(\*)(void) fillStableDelay )**

This function sets MCG to some target mode defined by the configure structure, if cannot switch to target mode directly, this function will choose the proper path.

## Function Documentation

### 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.

### 55.6.15 `clock_manager_error_code_t CLOCK_SYS_OscInit ( uint32_t instance, osc_user_config_t * config )`

This function initializes OSC according to board configuration.

### Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>instance</i> | Instance of the OSC.                        |
| <i>config</i>   | Pointer to the OSC configuration structure. |

### Returns

kClockManagerSuccess on success.

### 55.6.16 `void CLOCK_SYS_OscDeinit ( uint32_t instance )`

This function deinitializes OSC.

### Parameters

|                 |                      |
|-----------------|----------------------|
| <i>instance</i> | Instance of the OSC. |
|-----------------|----------------------|

### 55.6.17 `void CLOCK_SYS_SetOscerConfiguration ( uint32_t instance, oscer_config_t const * config )`

This function configures the OSCERCLK, including whether OSCERCLK is enable or not in normal mode and stop mode.

Parameters

|                 |                                                  |
|-----------------|--------------------------------------------------|
| <i>instance</i> | Instance of the OSC.                             |
| <i>config</i>   | Pointer to the OSCERCLK configuration structure. |

**55.6.18 static void CLOCK\_SYS\_SetOutDiv1 ( uint8\_t *outdiv1* ) [inline], [static]**

This function sets divide value OUTDIV1.

Parameters

|                |                     |
|----------------|---------------------|
| <i>outdiv1</i> | Outdivider1 setting |
|----------------|---------------------|

**55.6.19 static uint8\_t CLOCK\_SYS\_GetOutDiv1 ( void ) [inline], [static]**

This function gets divide value OUTDIV1.

Returns

Outdivider1 setting

**55.6.20 static void CLOCK\_SYS\_SetOutDiv2 ( uint8\_t *outdiv2* ) [inline], [static]**

This function sets divide value OUTDIV2.

Parameters

|                |                     |
|----------------|---------------------|
| <i>outdiv2</i> | Outdivider2 setting |
|----------------|---------------------|

**55.6.21 static uint8\_t CLOCK\_SYS\_GetOutDiv2 ( void ) [inline], [static]**

This function gets divide value OUTDIV2.

Returns

Outdivider2 setting

---

## Function Documentation

**55.6.22** `static void CLOCK_SYS_SetOutDiv4 ( uint8_t outdiv4 ) [inline],  
[static]`

This function sets divide value OUTDIV4.



Parameters

|                |                     |
|----------------|---------------------|
| <i>outdiv4</i> | Outdivider4 setting |
|----------------|---------------------|

**55.6.23 static uint8\_t CLOCK\_SYS\_GetOutDiv4 ( void ) [inline], [static]**

This function gets divide value OUTDIV4.

Returns

Outdivider4 setting

**55.6.24 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 |

**55.6.25 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 |

## Function Documentation

|                |                     |
|----------------|---------------------|
| <i>outdiv3</i> | Outdivider3 setting |
| <i>outdiv4</i> | Outdivider4 setting |

### 55.6.26 `uint32_t CLOCK_SYS_GetPIIFllClockFreq ( void )`

This function gets the frequency of the MCGPLLCLK/MCGFLLCLK/IRC48MCLK.

Returns

Current clock frequency.

### 55.6.27 `static void CLOCK_SYS_SetPIIFllSel ( clock_pllfl_sel_t setting ) [inline], [static]`

This function sets the selection of the high frequency clock for various peripheral clocking options

Parameters

|                |                   |
|----------------|-------------------|
| <i>setting</i> | The value to set. |
|----------------|-------------------|

### 55.6.28 `static clock_pllfl_sel_t CLOCK_SYS_GetPIIFllSel ( void ) [inline], [static]`

This function gets the selection of the high frequency clock for various peripheral clocking options

Returns

Current selection.

### 55.6.29 `static uint32_t CLOCK_SYS_GetFixedFreqClockFreq ( void ) [inline], [static]`

This function gets the MCG fixed frequency clock (MCGFFCLK) frequency.

Returns

Current clock frequency.

**55.6.30 static uint32\_t CLOCK\_SYS\_GetInternalRefClockFreq ( void )  
[inline], [static]**

This function gets the internal reference clock frequency.

Returns

Current clock frequency.

**55.6.31 uint32\_t CLOCK\_SYS\_GetExternalRefClock32kFreq ( void )**

This function gets the external reference (ERCLK32K) clock frequency.

Returns

Current frequency.

**55.6.32 static void CLOCK\_SYS\_SetExternalRefClock32kSrc ( clock\_er32k\_src\_t  
src ) [inline], [static]**

This function sets the clock selection of ERCLK32K.

Parameters

|            |               |
|------------|---------------|
| <i>src</i> | clock source. |
|------------|---------------|

**55.6.33 static clock\_er32k\_src\_t CLOCK\_SYS\_GetExternalRefClock32kSrc ( void  
) [inline], [static]**

This function gets the clock selection of ERCLK32K.

Returns

Current selection.

**55.6.34 uint32\_t CLOCK\_SYS\_GetOsc0ExternalRefClockFreq ( void )**

Gets the frequency.

This function gets the OSC0 external reference frequency.

## Function Documentation

Returns

Current frequency.

### 55.6.35 `uint32_t CLOCK_SYS_GetOsc0ExternalRefClockUndivFreq ( void )`

This function gets the undivided OSC0 external reference frequency.

Returns

Current frequency.

### 55.6.36 `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.

### 55.6.37 `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.

**55.6.38** `static void CLOCK_SYS_SetTraceSrc ( uint32_t instance, clock_trace_src_t src ) [inline], [static]`

This function sets the debug trace clock source.

## Function Documentation

Parameters

|                 |                           |
|-----------------|---------------------------|
| <i>instance</i> | module device instance.   |
| <i>src</i>      | debug trace clock source. |

### 55.6.39 uint32\_t CLOCK\_SYS\_GetTraceFreq ( uint32\_t *instance* )

This function gets the debug trace clock frequency.

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.

### 55.6.40 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.

**55.6.41** `uint32_t CLOCK_SYS_GetLptmrFreq ( uint32_t instance,  
clock_lptmr_src_t lptmrSrc )`

This function gets the LPTMRx pre-scaler/glitch filter clock frequency.

## Function Documentation

### Parameters

|                 |                                |
|-----------------|--------------------------------|
| <i>instance</i> | module device instance         |
| <i>lptmrSrc</i> | LPTMRx clock source selection. |

### Returns

Current frequency.

### 55.6.42 `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

### 55.6.43 `static uint32_t CLOCK_SYS_GetFtfFreq ( uint32_t instance ) [inline], [static]`

(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

(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

#### 55.6.44 **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

---

## Function Documentation

**55.6.45** `static uint32_t CLOCK_SYS_GetCmpFreq ( uint32_t instance )`  
`[inline], [static]`

This function gets the clock frequency for CMP module.

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

**55.6.46 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

**55.6.47 static uint32\_t CLOCK\_SYS\_GetPdbFreq ( uint32\_t *instance* ) [inline], [static]**

This function gets the clock frequency for PDB module.

Parameters

---

## Function Documentation

|                 |                        |
|-----------------|------------------------|
| <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

### 55.6.48 **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

### 55.6.49 **static uint32\_t CLOCK\_SYS\_GetSpiFreq ( uint32\_t *instance* ) [inline], [static]**

Gets the clock frequency for USB FS OTG module.

This function gets the clock frequency for SPI 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

Gets the clock frequency for SPI module

This function gets the clock frequency for SPI module.

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

**55.6.50** `static uint32_t CLOCK_SYS_GetI2cFreq ( uint32_t instance ) [inline], [static]`

This function gets the clock frequency for I2C module.

## Function Documentation

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

### 55.6.51 `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.

### 55.6.52 `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.

**55.6.53 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.

**55.6.54 static uint32\_t CLOCK\_SYS\_GetFtmSystemClockFreq ( uint32\_t *instance* )  
[inline], [static]**

This function gets the FTM's system clock frequency.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

Current frequency.

**55.6.55 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.

**55.6.56** `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.

**55.6.57** `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. |

**55.6.58** `uint32_t CLOCK_SYS_GetUartFreq ( uint32_t instance ) [inline]`

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 UART clock frequency.

Parameters

|                 |              |
|-----------------|--------------|
| <i>instance</i> | IP instance. |
|-----------------|--------------|

Returns

Current frequency.

This function gets the clock frequency for UART module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module

**55.6.59 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

#### 55.6.60 **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 |
|-----------------|------------------------|

#### 55.6.61 **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 |
|-----------------|------------------------|

#### 55.6.62 **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)

**55.6.63** `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 |
|-----------------|------------------------|

### 55.6.64 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 |
|-----------------|------------------------|

### 55.6.65 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)

### 55.6.66 void CLOCK\_SYS\_EnablePortClock ( uint32\_t *instance* )

This function enables the clock for PORT module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.67 void CLOCK\_SYS\_DisablePortClock ( uint32\_t *instance* )

This function disables the clock for PORT module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.68 **bool** CLOCK\_SYS\_GetPortGateCmd ( uint32\_t *instance* )

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)

### 55.6.69 **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 |
|-----------------|------------------------|

### 55.6.70 **static void** CLOCK\_SYS\_DisableEwmClock ( uint32\_t *instance* ) [inline], [static]

This function disables the clock for EWM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.71 **static bool** CLOCK\_SYS\_GetEwmGateCmd ( uint32\_t *instance* ) [inline], [static]

This function will get the clock gate state for EWM module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.72 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 |
|-----------------|------------------------|

**55.6.73 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 |
|-----------------|------------------------|

**55.6.74 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)

**55.6.75** `static void CLOCK_SYS_EnableCrcClock ( uint32_t instance ) [inline], [static]`

This function enables the clock for CRC module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.76 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 |
|-----------------|------------------------|

**55.6.77 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)

**55.6.78 void CLOCK\_SYS\_EnableAdcClock ( uint32\_t *instance* ) [inline]**

This function enables the clock for ADC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function enables the clock for ADC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|



**55.6.79 void CLOCK\_SYS\_DisableAdcClock ( uint32\_t *instance* ) [inline]**

This function disables the clock for ADC module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function disables the clock for ADC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.80 **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 will get the clock gate state for ADC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 55.6.81 **static void** CLOCK\_SYS\_EnableCmpClock ( uint32\_t *instance* ) [inline], [static]

This function enables the clock for CMP module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.82** `static void CLOCK_SYS_DisableCmpClock ( uint32_t instance )`  
`[inline], [static]`

This function disables the clock for CMP module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.83 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)

### 55.6.84 void CLOCK\_SYS\_EnableDacClock ( uint32\_t *instance* ) [inline]

This function enables the clock for DAC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function enables the clock for DAC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.85 void CLOCK\_SYS\_DisableDacClock ( uint32\_t *instance* ) [inline]

This function disables the clock for DAC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function disables the clock for DAC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.86 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 will get the clock gate state for DAC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.87 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 |
|-----------------|------------------------|

**55.6.88 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 |
|-----------------|------------------------|

**55.6.89 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)

**55.6.90 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 |
|-----------------|------------------------|

**55.6.91 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 |
|-----------------|------------------------|

**55.6.92 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)

**55.6.93 void CLOCK\_SYS\_EnableFtmClock ( uint32\_t *instance* ) [inline]**

This function enables the clock for FTM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function enables the clock for FTM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.94 void CLOCK\_SYS\_DisableFtmClock ( uint32\_t *instance* ) [inline]**

This function disables the clock for FTM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function disables the clock for FTM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.95 bool CLOCK\_SYS\_GetFtmGateCmd ( uint32\_t *instance* ) [inline]**

This function will get the clock gate state for FTM module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

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)

### 55.6.96 **static void CLOCK\_SYS\_EnablePitClock ( uint32\_t *instance* ) [inline], [static]**

This function enables the clock for PIT module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.97 **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 |
|-----------------|------------------------|

### 55.6.98 **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)

**55.6.99 static void CLOCK\_SYS\_EnableLptmrClock ( uint32\_t *instance* )  
[inline], [static]**

Enable the clock for LPTMR module.

This function enables the clock for LPTIMER module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function enables the clock for LPTMR module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.100 static void CLOCK\_SYS\_DisableLptmrClock ( uint32\_t *instance* )  
[inline], [static]**

Disable the clock for LPTMR module.

This function disables the clock for LPTIMER module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function disables the clock for LPTMR module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.101 static bool CLOCK\_SYS\_GetLptmrGateCmd ( uint32\_t *instance* )  
[inline], [static]**

Get the the clock gate state for LPTMR module.

---

## Function Documentation

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)

This function will get the clock gate state for LPTMR module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 55.6.102 void CLOCK\_SYS\_EnableSpiClock ( uint32\_t *instance* ) [inline]

Enable the clock for USBFS module.

This function enables the clock for SPI module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function enables the clock for USBFS module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Enable the clock for SPI module.

This function enables the clock for SPI module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function enables the clock for SPI module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.103 void CLOCK\_SYS\_DisableSpiClock ( uint32\_t *instance* ) [inline]

This function disables the clock for SPI module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function disables the clock for SPI module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.104 bool CLOCK\_SYS\_GetSpiGateCmd ( uint32\_t *instance* ) [inline]

This function will get the clock gate state for SPI 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 SPI module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 55.6.105 void CLOCK\_SYS\_EnableI2cClock ( uint32\_t *instance* ) [inline]

This function enables the clock for I2C module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function enables the clock for I2C module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.106 void CLOCK\_SYS\_DisableI2cClock ( uint32\_t *instance* ) [inline]

This function disables the clock for I2C module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function disables the clock for I2C module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.107 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 will get the clock gate state for I2C module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

---

## Function Documentation

**55.6.108** void `CLOCK_SYS_EnableUartClock` ( uint32\_t *instance* ) [inline]

This function enables the clock for UART module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function enables the clock for UART module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.109 void CLOCK\_SYS\_DisableUartClock ( uint32\_t *instance* ) [inline]

This function disables the clock for UART module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function disables the clock for UART module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.110 bool CLOCK\_SYS\_GetUartGateCmd ( uint32\_t *instance* ) [inline]

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)

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)

---

## Function Documentation

**55.6.111** `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.          |

**55.6.112 static void CLOCK\_SYS\_SetOutDiv3 ( uint8\_t *outdiv3* ) [inline], [static]**

This function sets divide value OUTDIV3.

Parameters

|                |                     |
|----------------|---------------------|
| <i>outdiv3</i> | Outdivider3 setting |
|----------------|---------------------|

**55.6.113 static uint8\_t CLOCK\_SYS\_GetOutDiv3 ( void ) [inline], [static]**

This function gets divide value OUTDIV3.

Returns

Outdivider3 setting

**55.6.114 uint32\_t CLOCK\_SYS\_GetFlexbusFreq ( void )**

This function gets the flexbus clock frequency.

Returns

Current flexbus clock frequency.

**55.6.115 static uint32\_t CLOCK\_SYS\_GetRtcFreq ( uint32\_t *instance* ) [inline], [static]**

This function gets the RTC input clock frequency.

## Function Documentation

### Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### Returns

Current frequency.

### 55.6.116 `uint32_t CLOCK_SYS_GetRtcOutFreq ( void )`

This function gets the frequency of RTC\_CLKOUT.

### Returns

Current frequency.

### 55.6.117 `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.

### 55.6.118 `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. |
|------------|---------------------------|

### 55.6.119 `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.

**55.6.120 static void CLOCK\_SYS\_SetEnetRmiiSrc ( uint32\_t *instance*, clock\_rmii\_src\_t *rmiiSrc* ) [inline], [static]**

This function sets the ethernet RMI clock source.

Parameters

|                 |                         |
|-----------------|-------------------------|
| <i>instance</i> | module device instance. |
| <i>rmiiSrc</i>  | RMI clock source.       |

**55.6.121 uint32\_t CLOCK\_SYS\_GetEnetRmiiFreq ( uint32\_t *instance* )**

This function gets the ethernet RMI clock frequency.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

Current frequency.

**55.6.122 uint32\_t CLOCK\_SYS\_GetFlexcanFreq ( uint32\_t *instance*, clock\_flexcan\_src\_t *flexcanSrc* )**

This function gets the FLEXCAN clock frequency.

## Function Documentation

### Parameters

|                   |                        |
|-------------------|------------------------|
| <i>instance</i>   | module device instance |
| <i>flexcanSrc</i> | clock source           |

### Returns

Current frequency.

**55.6.123** `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. |

**55.6.124** `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.

**55.6.125** `uint32_t CLOCK_SYS_GetEnetTimeStampFreq ( uint32_t instance )`

This function gets the ethernet timestamp clock frequency.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

Current frequency.

**55.6.126 static uint32\_t CLOCK\_SYS\_GetCmtFreq ( uint32\_t *instance* )  
[inline], [static]**

This function gets the clock frequency for CMT module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module

**55.6.127 uint32\_t CLOCK\_SYS\_GetSdhcFreq ( uint32\_t *instance* )**

This function gets the clock frequency for SDHC.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module.

**55.6.128 static void CLOCK\_SYS\_SetSdhcSrc ( uint32\_t *instance*,  
clock\_sdhc\_src\_t *setting* ) [inline], [static]**

This function sets the SDHC clock source selection.

## Function Documentation

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>instance</i> | IP instance.      |
| <i>setting</i>  | The value to set. |

### 55.6.129 `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.

### 55.6.130 `uint32_t CLOCK_SYS_GetSaiFreq ( uint32_t instance, clock_sai_src_t saiSrc )`

This function gets the clock frequency for SAI.

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.

### 55.6.131 `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

**55.6.132 static void CLOCK\_SYS\_EnableMpuClock ( uint32\_t *instance* )  
[inline], [static]**

This function enables the clock for MPU module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.133 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 |
|-----------------|------------------------|

**55.6.134 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)

---

## Function Documentation

**55.6.135** `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 |
|-----------------|------------------------|

**55.6.136 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 |
|-----------------|------------------------|

**55.6.137 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)

**55.6.138 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 |
|-----------------|------------------------|

**55.6.139 static void CLOCK\_SYS\_DisableRngaClock ( uint32\_t *instance* )  
[inline], [static]**

This function disables the clock for RNGA module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.140 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)

**55.6.141 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 |
|-----------------|------------------------|

**55.6.142 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 |
|-----------------|------------------------|

**55.6.143 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)

**55.6.144 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 |
|-----------------|------------------------|

**55.6.145 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 |
|-----------------|------------------------|

**55.6.146 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)

---

## Function Documentation

**55.6.147** `static void CLOCK_SYS_EnableRtcClock ( uint32_t instance )`  
`[inline], [static]`

This function enables the clock for RTC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.148 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 |
|-----------------|------------------------|

**55.6.149 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)

**55.6.150 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 |
|-----------------|------------------------|

**55.6.151 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 |
|-----------------|------------------------|

### 55.6.152 **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)

### 55.6.153 **static void CLOCK\_SYS\_EnableFlexcanClock ( uint32\_t *instance* )** **[inline]**

This function enables the clock for FLEXCAN module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function enables the clock for FLEXCAN module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.154 **static void CLOCK\_SYS\_DisableFlexcanClock ( uint32\_t *instance* )** **[inline]**

This function disables the clock for FLEXCAN module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function disables the clock for FLEXCAN module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.155 static bool CLOCK\_SYS\_GetFlexcanGateCmd ( uint32\_t *instance* )  
[inline]**

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)

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)

**55.6.156 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 |
|-----------------|------------------------|

**55.6.157 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 |
|-----------------|------------------------|

**55.6.158 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)

**55.6.159 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 |
|-----------------|------------------------|

**55.6.160 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 |
|-----------------|------------------------|

**55.6.161 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)

**55.6.162 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 |
|-----------------|------------------------|

**55.6.163 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 |
|-----------------|------------------------|

**55.6.164 static bool CLOCK\_SYS\_GetLlwuGateCmd ( uint32\_t *instance* )  
[inline], [static]**

This function will get the clock gate state for LLWU module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.165 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.          |

**55.6.166 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.          |

**55.6.167 uint32\_t CLOCK\_SYS\_GetUsbfsFreq ( uint32\_t *instance* )**

Gets the clock frequency for USBFS 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

This function gets the clock frequency for USBFS module.

## Function Documentation

### Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### Returns

freq clock frequency for this module

**55.6.168** `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 \* [ (USBFSFRAC+1) / (USBFSDIV+1) ]

### Parameters

|                 |                         |
|-----------------|-------------------------|
| <i>instance</i> | USB FS module instance. |
| <i>usbdiv</i>   | Value of USBFSDIV.      |
| <i>usbfrac</i>  | Value of USBFSFRAC.     |

**55.6.169** `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 \* [ (USBFSFRAC+1) / (USBFSDIV+1) ]

### Parameters

|                 |                         |
|-----------------|-------------------------|
| <i>instance</i> | USB FS module instance. |
| <i>usbdiv</i>   | Value of USBFSDIV.      |
| <i>usbfrac</i>  | Value of USBFSFRAC.     |

**55.6.170** `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 |
|-----------------|------------------------|

**55.6.171 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 |
|-----------------|------------------------|

**55.6.172 static bool CLOCK\_SYS\_GetUsbfsGateCmd ( uint32\_t *instance* )  
[inline], [static]**

Get the the clock gate state for USBFS module.

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)

This function will get the clock gate state for USBFS module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.173 static void CLOCK\_SYS\_EnableUsbdcdClock ( uint32\_t *instance* )  
[inline], [static]**

This function enables the clock for USBDCD module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.174 static void CLOCK\_SYS\_DisableUsbdcdClock ( uint32\_t *instance* )  
[inline], [static]**

This function disables the clock for USBDCD module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.175 static bool CLOCK\_SYS\_GetUsbdcdGateCmd ( uint32\_t *instance* )  
[inline], [static]**

This function will get the clock gate state for USBDCD module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.176 static clock\_usbfs\_src\_t CLOCK\_SYS\_GetUsbfsSrc ( uint32\_t *instance* )  
[inline], [static]**

Gets the clock source for USBFS module.

This function gets the clock source for USB FS OTG module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

Clock source.

This function gets the clock source for USBFS module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

source Clock source for this module.

**55.6.177 static void CLOCK\_SYS\_SetUsbfsSrc ( uint32\_t *instance*, clock\_usbfs\_src\_t *usbfsSrc* ) [inline], [static]**

Sets the clock source for USBFS module.

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. |

This function sets the clock source for USBFS module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
| <i>usbfsSrc</i> | Clock source to set.   |

**55.6.178 void CLOCK\_SYS\_Osc0Deinit ( void )**

This function deinitializes OSC0.

**55.6.179 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>        | Frequcney value.          |

---

## Function Documentation

**55.6.180** `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.

This function gets the clock source for LPUART module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

source Clock source for this module.

### 55.6.181 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.          |

This function sets the clock source for LPUART module.

Parameters

|                  |                        |
|------------------|------------------------|
| <i>instance</i>  | module device instance |
| <i>lpuartSrc</i> | Clock source to set.   |

### 55.6.182 uint32\_t CLOCK\_SYS\_GetLpuartFreq ( uint32\_t *instance* )

This function gets the clock frequency for LPUART module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module

### 55.6.183 **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 |
|-----------------|------------------------|

### 55.6.184 **static void CLOCK\_SYS\_DisableLpuartClock ( uint32\_t *instance* ) [inline], [static]**

Disable the clock for UART module.

This function disables the clock for LPUART module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

This function disables the clock for UART module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.185 **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.

## Function Documentation

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)

### 55.6.186 uint32\_t CLOCK\_SYS\_GetPIIFllDivClockFreq ( void )

This function gets the frequency of the PLL/FLL clock divided by the fractional divider configured by SIM\_CLKDIV3[PLLFLFRAC, PLLFLLDIV].

Returns

Current clock frequency.

### 55.6.187 uint32\_t CLOCK\_SYS\_GetTpmFreq ( uint32\_t instance )

Gets the clock frequency for TPM module.

This function gets the TPM clock frequency.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

Current frequency.

This function gets the clock frequency for TPM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module

**55.6.188 static void CLOCK\_SYS\_SetTpmSrc ( uint32\_t *instance*, clock\_tpm\_src\_t *tpmSrc* ) [inline], [static]**

Sets the clock source for TPM module.

This function sets the TPM clock source selection.

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>instance</i> | IP instance.      |
| <i>tpmSrc</i>   | The value to set. |

This function sets the clock source for TPM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
| <i>tpmSrc</i>   | Clock source to set.   |

**55.6.189 static clock\_tpm\_src\_t CLOCK\_SYS\_GetTpmSrc ( uint32\_t *instance* ) [inline], [static]**

Gets the clock source for TPM module.

This function gets the TPM clock source selection.

Parameters

|                 |              |
|-----------------|--------------|
| <i>instance</i> | IP instance. |
|-----------------|--------------|

Returns

Current selection.

This function gets the clock source for TPM module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

source Clock source for this module.

### 55.6.190 `uint32_t CLOCK_SYS_GetTpmExternalFreq ( uint32_t instance )`

Gets the external clock frequency for TPM module.

This function gets the TPM external clock source frequency.

Parameters

|                 |              |
|-----------------|--------------|
| <i>instance</i> | IP instance. |
|-----------------|--------------|

Returns

TPM external clock frequency.

This function gets the external clock frequency for TPM module.

Parameters

|                 |                         |
|-----------------|-------------------------|
| <i>instance</i> | module device instance. |
|-----------------|-------------------------|

Returns

freq External clock frequency.

### 55.6.191 `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. |

**55.6.192** `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.

**55.6.193** `static clock_usbhs_slowclk_src_t CLOCK_SYS_GetUsbhsSlowClockSrc ( uint32_t instance ) [inline], [static]`

This function gets the slow clock source for USB HS/USB PHY module, used to detect wakeup and resume events.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

Clock source.

**55.6.194** `static void CLOCK_SYS_SetUsbhsSlowClockSrc ( uint32_t instance, clock_usbhs_slowclk_src_t usbhsSrc ) [inline], [static]`

This function sets the clock source for USB HS/USB PHY module, used to detect wakeup and resume events.

## Function Documentation

Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>instance</i> | module device instance.      |
| <i>usbhsSrc</i> | USB FS clock source setting. |

### 55.6.195 `uint32_t CLOCK_SYS_GetUsbhsSlowClockFreq ( uint32_t instance )`

This function gets the clock frequency for USB HS/USB PHY module, used to detect wakeup and resume events.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module

### 55.6.196 `static void CLOCK_SYS_EnableSdramcClock ( uint32_t instance )` `[inline], [static]`

This function enables the clock for SDRAMC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.197 `static void CLOCK_SYS_DisableSdramcClock ( uint32_t instance )` `[inline], [static]`

This function disables the clock for SDRAMC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.198 `static bool CLOCK_SYS_GetSdramcGateCmd ( uint32_t instance )` `[inline], [static]`

This function will get the clock gate state for SDRAMC module.



Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.199 void CLOCK\_SYS\_EnableTpmClock ( uint32\_t *instance* )**

This function enables the clock for TPM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.200 void CLOCK\_SYS\_DisableTpmClock ( uint32\_t *instance* )**

This function disables the clock for TPM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.201 bool CLOCK\_SYS\_GetTpmGateCmd ( uint32\_t *instance* )**

This function will get the clock gate state for TPM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.202 static void CLOCK\_SYS\_EnableUsbhsClock ( uint32\_t *instance* )  
[inline], [static]**

This function enables the clock for USBHS module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.203 static void CLOCK\_SYS\_DisableUsbhsClock ( uint32\_t *instance* )**  
**[inline], [static]**

This function disables the clock for USBHS module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.204 static bool CLOCK\_SYS\_GetUsbhsGateCmd ( uint32\_t *instance* )**  
**[inline], [static]**

This function will get the clock gate state for USBHS module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.205 static void CLOCK\_SYS\_EnableUsbphyClock ( uint32\_t *instance* )**  
**[inline], [static]**

This function enables the clock for USBPHY module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.206 static void CLOCK\_SYS\_DisableUsbphyClock ( uint32\_t *instance* )**  
**[inline], [static]**

This function disables the clock for USBPHY module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.207 static bool CLOCK\_SYS\_GetUsbphyGateCmd ( uint32\_t *instance* )  
[inline], [static]**

This function will get the clock gate state for USBPHY module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.208 static void CLOCK\_SYS\_EnableUsbhsdcdClock ( uint32\_t *instance* )  
[inline], [static]**

This function enables the clock for USBHSDCD module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.209 static void CLOCK\_SYS\_DisableUsbhsdcdClock ( uint32\_t *instance* )  
[inline], [static]**

This function disables the clock for USBHSDCD module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.210 static bool CLOCK\_SYS\_GetUsbhsdcdGateCmd ( uint32\_t *instance* )  
[inline], [static]**

This function will get the clock gate state for USBHSDCD module.

## Function Documentation

### Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### Returns

state true - ungated(Enabled), false - gated (Disabled)

### 55.6.211 uint32\_t CLOCK\_SYS\_GetFIIClockFreq ( void )

This function gets the frequency of the MCGFLLCLK.

### Returns

Current clock frequency.

### 55.6.212 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.

### 55.6.213 uint32\_t CLOCK\_SYS\_GetLpsciFreq ( 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

**55.6.214** `static void CLOCK_SYS_SetLpSciSrc ( uint32_t instance,  
clock_lpSci_src_t lpSciSrc ) [inline], [static]`

This function sets the LPSCI clock source selection.

## Function Documentation

Parameters

|                 |                   |
|-----------------|-------------------|
| <i>instance</i> | IP instance.      |
| <i>lpsciSrc</i> | The value to set. |

### 55.6.215 `static clock_lpsci_src_t CLOCK_SYS_GetLpsciSrc ( uint32_t instance ) [inline], [static]`

This function gets the LPSCI clock source selection.

Parameters

|                 |              |
|-----------------|--------------|
| <i>instance</i> | IP instance. |
|-----------------|--------------|

Returns

Current selection.

### 55.6.216 `void CLOCK_SYS_EnableLpsciClock ( uint32_t instance )`

This function enables the clock for LPSCI module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.217 `void CLOCK_SYS_DisableLpsciClock ( uint32_t instance )`

This function disables the clock for LPSCI module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.218 `bool CLOCK_SYS_GetLpsciGateCmd ( 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)

**55.6.219 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>        | Frequcney value.          |

**55.6.220 uint32\_t CLOCK\_SYS\_GetCopFreq ( void )**

This function gets the COP clock frequency.

Returns

Current frequency.

**55.6.221 static void CLOCK\_SYS\_SetCopSrc ( clock\_cop\_src\_t *copSrc* ) [inline], [static]**

This function sets the COP clock source selection.

Parameters

|               |                   |
|---------------|-------------------|
| <i>copSrc</i> | The value to set. |
|---------------|-------------------|

**55.6.222 static clock\_cop\_src\_t CLOCK\_SYS\_GetCopSrc ( void ) [inline], [static]**

This function gets the COP clock source selection.

## Function Documentation

Returns

Current selection.

### 55.6.223 uint32\_t CLOCK\_SYS\_GetFlexioFreq ( uint32\_t *instance* )

Gets the clock frequency for FLEXIO module.

This function gets the FLEXIO clock frequency.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

Current frequency.

This function gets the clock frequency for FLEXIO module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module

### 55.6.224 static void CLOCK\_SYS\_SetFlexioSrc ( uint32\_t *instance*, clock\_flexio\_src\_t *flexioSrc* ) [inline], [static]

Sets the clock source for FLEXIO module.

This function sets the FLEXIO clock source selection.

Parameters

|                  |                   |
|------------------|-------------------|
| <i>instance</i>  | IP instance.      |
| <i>flexioSrc</i> | The value to set. |

This function sets the clock source for FLEXIO module.



Parameters

|                  |                        |
|------------------|------------------------|
| <i>instance</i>  | module device instance |
| <i>flexioSrc</i> | Clock source to set.   |

**55.6.225 static clock\_flexio\_src\_t CLOCK\_SYS\_GetFlexioSrc ( uint32\_t *instance* )  
[inline], [static]**

Gets the clock source for FLEXIO module.

This function gets the FLEXIO clock source selection.

Parameters

|                 |              |
|-----------------|--------------|
| <i>instance</i> | IP instance. |
|-----------------|--------------|

Returns

Current selection.

This function gets the clock source for FLEXIO module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

source Clock source for this module.

**55.6.226 static void CLOCK\_SYS\_EnableFlexioClock ( uint32\_t *instance* )  
[inline], [static]**

This function enables the clock for FLEXIO module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.227 static void CLOCK\_SYS\_DisableFlexioClock ( uint32\_t *instance* )  
[inline], [static]**

This function disables the clock for FLEXIO module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.228 static bool CLOCK\_SYS\_GetFlexioGateCmd ( uint32\_t *instance* ) [inline], [static]

Get the the clock gate state for FLEXIO module.

This function will get the clock gate state for FLEXIO module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 55.6.229 void CLOCK\_SYS\_EnableXrdcClock ( uint32\_t *instance* )

This function enables the clock for XRDC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.230 void CLOCK\_SYS\_DisableXrdcClock ( uint32\_t *instance* )

This function disables the clock for XRDC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.231 bool CLOCK\_SYS\_GetXrdcGateCmd ( uint32\_t *instance* )

This function will get the clock gate state for XRDC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.232 void CLOCK\_SYS\_EnableSemaClock ( uint32\_t *instance* )**

This function enables the clock for SEMA module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.233 void CLOCK\_SYS\_DisableSemaClock ( uint32\_t *instance* )**

This function disables the clock for SEMA module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.234 bool CLOCK\_SYS\_GetSemaGateCmd ( uint32\_t *instance* )**

This function will get the clock gate state for SEMA module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.235 void CLOCK\_SYS\_EnableFlashClock ( uint32\_t *instance* )**

This function enables the clock for FLASH module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.236 void CLOCK\_SYS\_DisableFlashClock ( uint32\_t *instance* )

This function disables the clock for FLASH module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.237 bool CLOCK\_SYS\_GetFlashGateCmd ( uint32\_t *instance* )

This function will get the clock gate state for FLASH module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 55.6.238 void CLOCK\_SYS\_EnableMuClock ( uint32\_t *instance* )

This function enables the clock for MU module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.239 void CLOCK\_SYS\_DisableMuClock ( uint32\_t *instance* )

This function disables the clock for MU module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.240 bool CLOCK\_SYS\_GetMuGateCmd ( uint32\_t *instance* )**

This function will get the clock gate state for MU module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.241 void CLOCK\_SYS\_EnableIntmuxClock ( uint32\_t *instance* )**

This function enables the clock for INTMUX module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.242 void CLOCK\_SYS\_DisableIntmuxClock ( uint32\_t *instance* )**

This function disables the clock for INTMUX module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.243 bool CLOCK\_SYS\_GetIntmuxGateCmd ( uint32\_t *instance* )**

This function will get the clock gate state for INTMUX module.

## Function Documentation

### Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### Returns

state true - ungated(Enabled), false - gated (Disabled)

### 55.6.244 clock\_pit\_src\_t CLOCK\_SYS\_GetPitSrc ( uint32\_t *instance* )

This function gets the clock source for PIT module.

### Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### Returns

source Clock source for this module.

### 55.6.245 void CLOCK\_SYS\_SetPitSrc ( uint32\_t *instance*, clock\_pit\_src\_t *pitSrc* )

This function sets the clock source for PIT module.

### Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
| <i>pitSrc</i>   | Clock source to set.   |

### 55.6.246 void CLOCK\_SYS\_EnableLpspiClock ( uint32\_t *instance* )

This function enables the clock for LPSPI module.

### Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.247 void CLOCK\_SYS\_DisableLpspiClock ( uint32\_t *instance* )

This function disables the clock for LPSPI module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.248 **bool CLOCK\_SYS\_GetLpspiGateCmd ( uint32\_t *instance* )**

This function will get the clock gate state for LPSPI module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 55.6.249 **clock\_lpspi\_src\_t CLOCK\_SYS\_GetLpspiSrc ( uint32\_t *instance* )**

This function gets the clock source for LPSPI module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

source Clock source for this module.

### 55.6.250 **void CLOCK\_SYS\_SetLpspiSrc ( uint32\_t *instance*, clock\_lpspi\_src\_t *lpspiSrc* )**

This function sets the clock source for LPSPI module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
| <i>lpspiSrc</i> | Clock source to set.   |

### 55.6.251 **uint32\_t CLOCK\_SYS\_GetLpspiFreq ( uint32\_t *instance* )**

This function gets the clock frequency for LPSPI module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module

### 55.6.252 void CLOCK\_SYS\_EnableLpi2cClock ( uint32\_t *instance* )

This function enables the clock for LPI2C module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.253 void CLOCK\_SYS\_DisableLpi2cClock ( uint32\_t *instance* )

This function disables the clock for LPI2C module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.254 bool CLOCK\_SYS\_GetLpi2cGateCmd ( uint32\_t *instance* )

This function will get the clock gate state for LPI2C module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 55.6.255 clock\_lpi2c\_src\_t CLOCK\_SYS\_GetLpi2cSrc ( uint32\_t *instance* )

This function gets the clock source for LPI2C module.



Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

source Clock source for this module.

**55.6.256 void CLOCK\_SYS\_SetLpi2cSrc ( uint32\_t *instance*, clock\_lpi2c\_src\_t *lpi2cSrc* )**

This function sets the clock source for LPI2C module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
| <i>lpi2cSrc</i> | Clock source to set.   |

**55.6.257 uint32\_t CLOCK\_SYS\_GetLpi2cFreq ( uint32\_t *instance* )**

This function gets the clock frequency for LPI2C module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module

**55.6.258 void CLOCK\_SYS\_EnableEvmsimClock ( uint32\_t *instance* )**

This function enables the clock for EVMSIM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.259 void CLOCK\_SYS\_DisableEvmsimClock ( uint32\_t *instance* )**

This function disables the clock for EVMSIM module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.260 **bool CLOCK\_SYS\_GetEvmsimGateCmd ( uint32\_t *instance* )**

This function will get the clock gate state for EVMSIM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 55.6.261 **clock\_evmsim\_src\_t CLOCK\_SYS\_GetEvmsimSrc ( uint32\_t *instance* )**

This function gets the clock source for EVMSIM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

source Clock source for this module.

### 55.6.262 **void CLOCK\_SYS\_SetEvmsimSrc ( uint32\_t *instance*, clock\_evmsim\_src\_t *evmsimSrc* )**

This function sets the clock source for EVMSIM module.

Parameters

|                  |                        |
|------------------|------------------------|
| <i>instance</i>  | module device instance |
| <i>evmsimSrc</i> | Clock source to set.   |

### 55.6.263 **uint32\_t CLOCK\_SYS\_GetEvmsimFreq ( uint32\_t *instance* )**

This function gets the clock frequency for EVMSIM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module

**55.6.264 void CLOCK\_SYS\_EnableAtxClock ( uint32\_t *instance* )**

This function enables the clock for ATX module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.265 void CLOCK\_SYS\_DisableAtxClock ( uint32\_t *instance* )**

This function disables the clock for ATX module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.266 bool CLOCK\_SYS\_GetAtxGateCmd ( uint32\_t *instance* )**

This function will get the clock gate state for ATX module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.267 void CLOCK\_SYS\_EnableTrngClock ( uint32\_t *instance* )**

This function enables the clock for TRNG module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.268 void CLOCK\_SYS\_DisableTrngClock ( uint32\_t *instance* )

This function disables the clock for TRNG module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.269 bool CLOCK\_SYS\_GetTrngGateCmd ( uint32\_t *instance* )

This function will get the clock gate state for TRNG module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

### 55.6.270 static void CLOCK\_SYS\_SetOutDiv5 ( uint8\_t *outdiv5* ) [inline], [static]

This function sets divide value OUTDIV5.

Parameters

|                |                     |
|----------------|---------------------|
| <i>outdiv5</i> | Outdivider5 setting |
|----------------|---------------------|

### 55.6.271 static uint8\_t CLOCK\_SYS\_GetOutDiv5 ( void ) [inline], [static]

This function gets divide value OUTDIV5.

Returns

Outdivider5 setting

**55.6.272 uint32\_t CLOCK\_SYS\_GetOutdiv5ClockFreq ( void )**

This function gets the OUTDIV5 output clock for ADC.

Returns

freq Current frequency.

**55.6.273 uint32\_t CLOCK\_SYS\_GetFastPeripheralClockFreq ( void )**

This function gets the fast peripheral clock frequency.

Returns

Current clock frequency.

**55.6.274 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

**55.6.275 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

**55.6.276 uint32\_t CLOCK\_SYS\_GetAdcFreq ( uint32\_t *instance* )**

This function gets the clock frequency for ADC module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module

### 55.6.277 uint32\_t CLOCK\_SYS\_GetPwmFreq ( uint32\_t *instance* )

This function gets the clock frequency for eFlexPWM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module

### 55.6.278 uint32\_t CLOCK\_SYS\_GetEncFreq ( uint32\_t *instance* )

This function gets the clock frequency for ENC module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module

### 55.6.279 uint32\_t CLOCK\_SYS\_GetXbarFreq ( uint32\_t *instance* )

This function gets the clock frequency for XBAR module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module

**55.6.280 uint32\_t CLOCK\_SYS\_GetAoiFreq ( uint32\_t *instance* )**

This function gets the clock frequency for AOI module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

freq clock frequency for this module

**55.6.281 uint32\_t CLOCK\_SYS\_GetFlexcanFreq ( uint8\_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.

**55.6.282 static void CLOCK\_SYS\_EnablePwmClock ( uint32\_t *instance* )  
[inline], [static]**

This function enables the clock for eFlexPWM module.

## Function Documentation

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.283 static void CLOCK\_SYS\_DisablePwmClock ( uint32\_t *instance* )**  
**[inline], [static]**

This function disables the clock for eFlexPWM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.284 static bool CLOCK\_SYS\_GetPwmGateCmd ( uint32\_t *instance* )**  
**[inline], [static]**

This function will get the clock gate state for eFlexPWM module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.285 static void CLOCK\_SYS\_EnableAoiClock ( uint32\_t *instance* )**  
**[inline], [static]**

This function enables the clock for AOI module.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.286 static void CLOCK\_SYS\_DisableAoiClock ( uint32\_t *instance* )**  
**[inline], [static]**

This function disables the clock for AOI module.



Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.287 static bool CLOCK\_SYS\_GetAoiGateCmd ( uint32\_t *instance* )  
[inline], [static]**

This function will get the clock gate state for AOI moudle.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

Returns

state true - ungated(Enabled), false - gated (Disabled)

**55.6.288 static void CLOCK\_SYS\_EnableXbarClock ( uint32\_t *instance* )  
[inline], [static]**

This function enables the clock for XBAR moudle.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.289 static void CLOCK\_SYS\_DisableXbarClock ( uint32\_t *instance* )  
[inline], [static]**

This function disables the clock for XBAR moudle.

Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

**55.6.290 static bool CLOCK\_SYS\_GetXbarGateCmd ( uint32\_t *instance* )  
[inline], [static]**

This function will get the clock gate state for XBAR moudle.

## Function Documentation

### Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### Returns

state true - ungated(Enabled), false - gated (Disabled)

### 55.6.291 static void CLOCK\_SYS\_EnableEncClock ( uint32\_t *instance* ) [inline], [static]

This function enables the clock for ENC mouldle.

### Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.292 static void CLOCK\_SYS\_DisableEncClock ( uint32\_t *instance* ) [inline], [static]

This function disables the clock for ENC mouldle.

### Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### 55.6.293 static bool CLOCK\_SYS\_GetEncGateCmd ( uint32\_t *instance* ) [inline], [static]

This function will get the clock gate state for ENC mouldle.

### Parameters

|                 |                        |
|-----------------|------------------------|
| <i>instance</i> | module device instance |
|-----------------|------------------------|

### Returns

state true - ungated(Enabled), false - gated (Disabled)

## **55.7 Variable Documentation**

**55.7.1 SIM\_Type\* const g\_simBase[]**

**55.7.2 MCG\_Type\* const g\_mcgBase[]**

**55.7.3 OSC\_Type\* const g\_oscBase[]**

## K02F12810

### 55.8 K02F12810

The data structure definition for K02F12810 clock manager.

#### 55.8.1 Overview

##### Files

- file [fsl\\_clock\\_MK02F12810.h](#)

##### Data Structures

- 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.*

##### Variables

- uint32\_t [g\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency.*

#### 55.8.2 Data Structure Documentation

##### 55.8.2.1 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.*

#### 55.8.2.1.0.10 Field Documentation

55.8.2.1.0.10.1 `clock_pllflr_sel_t sim_config_k02f12810_t::pllFlrSel`

55.8.2.1.0.10.2 `clock_er32k_src_t sim_config_k02f12810_t::er32kSrc`

55.8.2.1.0.10.3 `uint8_t sim_config_k02f12810_t::outdiv4`

#### 55.8.3 Macro Definition Documentation

55.8.3.1 `#define FTM_EXT_CLK_COUNT 2`

#### 55.8.4 Variable Documentation

55.8.4.1 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K10D10

### 55.9 K10D10

The data structure definition for K10D10 clock manager.

#### 55.9.1 Overview

##### Files

- file [fsl\\_clock\\_MK10D10.h](#)

##### Data Structures

- struct [sim\\_config\\_k10d10\\_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.*

##### 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\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency([FTM\\_CLK](#)).*

#### 55.9.2 Data Structure Documentation

##### 55.9.2.1 struct sim\_config\_k10d10\_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.*

#### 55.9.2.1.0.11 Field Documentation

55.9.2.1.0.11.1 `clock_pllfl_sel_t sim_config_k10d10_t::pllFlSel`

55.9.2.1.0.11.2 `clock_er32k_src_t sim_config_k10d10_t::er32kSrc`

55.9.2.1.0.11.3 `uint8_t sim_config_k10d10_t::outdiv4`

#### 55.9.3 Macro Definition Documentation

55.9.3.1 `#define ENET_EXT_CLK_COUNT 1`

55.9.3.2 `#define SDHC_EXT_CLK_COUNT 1`

55.9.3.3 `#define USB_EXT_CLK_COUNT 1`

55.9.3.4 `#define FTM_EXT_CLK_COUNT 2`

#### 55.9.4 Variable Documentation

55.9.4.1 `uint32_t g_sdhcClkInFreq[SDHC_EXT_CLK_COUNT]`

55.9.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K11DA5

### 55.10 K11DA5

The data structure definition for K11DA5 clock manager.

#### 55.10.1 Overview

##### Files

- file [fsl\\_clock\\_MK11DA5.h](#)

##### Data Structures

- struct [sim\\_config\\_k11da5\\_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).*

#### 55.10.2 Data Structure Documentation

##### 55.10.2.1 struct sim\_config\_k11da5\_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.*

#### 55.10.2.1.0.12 Field Documentation

55.10.2.1.0.12.1 `clock_pllfl_sel_t sim_config_k11da5_t::pllFlSel`

55.10.2.1.0.12.2 `clock_er32k_src_t sim_config_k11da5_t::er32kSrc`

55.10.2.1.0.12.3 `uint8_t sim_config_k11da5_t::outdiv4`

#### 55.10.3 Macro Definition Documentation

55.10.3.1 `#define USB_EXT_CLK_COUNT 1`

55.10.3.2 `#define FTM_EXT_CLK_COUNT 2`

55.10.3.3 `#define CLOCK_CONFIG_NUM 2`

55.10.3.4 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

55.10.3.5 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

#### 55.10.4 Variable Documentation

55.10.4.1 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

55.10.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K20D10

### 55.11 K20D10

The data structure definition for K20D10 clock manager.

#### 55.11.1 Overview

##### Files

- file [fsl\\_clock\\_MK20D10.h](#)

##### Data Structures

- struct [sim\\_config\\_k20d10\\_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.*

##### 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\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency([FTM\\_CLK](#)).*

#### 55.11.2 Data Structure Documentation

##### 55.11.2.1 struct [sim\\_config\\_k20d10\\_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.*

#### 55.11.2.1.0.13 Field Documentation

55.11.2.1.0.13.1 `clock_pllfl_sel_t sim_config_k20d10_t::pllFlSel`

55.11.2.1.0.13.2 `clock_er32k_src_t sim_config_k20d10_t::er32kSrc`

55.11.2.1.0.13.3 `uint8_t sim_config_k20d10_t::outdiv4`

#### 55.11.3 Macro Definition Documentation

55.11.3.1 `#define ENET_EXT_CLK_COUNT 1`

55.11.3.2 `#define SDHC_EXT_CLK_COUNT 1`

55.11.3.3 `#define USB_EXT_CLK_COUNT 1`

55.11.3.4 `#define FTM_EXT_CLK_COUNT 2`

#### 55.11.4 Variable Documentation

55.11.4.1 `uint32_t g_sdhcClkInFreq[SDHC_EXT_CLK_COUNT]`

55.11.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K21DA5

### 55.12 K21DA5

The data structure definition for K21DA5 clock manager.

#### 55.12.1 Overview

##### Files

- file [fsl\\_clock\\_MK21DA5.h](#)

##### Data Structures

- struct [sim\\_config\\_k21da5\\_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\\_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).*

#### 55.12.2 Data Structure Documentation

##### 55.12.2.1 struct sim\_config\_k21da5\_t

##### Data Fields

- [clock\\_pllfl\\_sel\\_t](#) [pllFlISel](#)

- *PLL/FLL/IRC48M selection.*  
clock\_er32k\_src\_t er32kSrc
- *ERCLK32K source selection.*  
uint8\_t outdiv4
- *OUTDIV setting.*

#### 55.12.2.1.0.14 Field Documentation

55.12.2.1.0.14.1 clock\_pllfl\_sel\_t sim\_config\_k21da5\_t::pllFlSel

55.12.2.1.0.14.2 clock\_er32k\_src\_t sim\_config\_k21da5\_t::er32kSrc

55.12.2.1.0.14.3 uint8\_t sim\_config\_k21da5\_t::outdiv4

#### 55.12.3 Macro Definition Documentation

55.12.3.1 #define SDHC\_EXT\_CLK\_COUNT 1

55.12.3.2 #define USB\_EXT\_CLK\_COUNT 1

55.12.3.3 #define FTM\_EXT\_CLK\_COUNT 2

55.12.3.4 #define CLOCK\_CONFIG\_NUM 2

55.12.3.5 #define CLOCK\_CONFIG\_INDEX\_FOR\_VLPR 0

55.12.3.6 #define CLOCK\_CONFIG\_INDEX\_FOR\_RUN 1

#### 55.12.4 Variable Documentation

55.12.4.1 uint32\_t g\_usbClkInFreq[USB\_EXT\_CLK\_COUNT]

55.12.4.2 uint32\_t g\_ftmClkFreq[FTM\_EXT\_CLK\_COUNT]

## K21FA12

### 55.13 K21FA12

The data structure definition for K21FA12 clock manager.

#### 55.13.1 Overview

##### Files

- file [fsl\\_clock\\_MK21FA12.h](#)

##### Data Structures

- struct [sim\\_config\\_k21fa12\\_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.*

##### 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).*

#### 55.13.2 Data Structure Documentation

##### 55.13.2.1 struct sim\_config\_k21fa12\_t

##### Data Fields

- [clock\\_pllfl\\_sel\\_t](#) [pllFlISel](#)  
*PLL/FLL/IRC48M selection.*
- [clock\\_er32k\\_src\\_t](#) [er32kSrc](#)  
*ERCLK32K source selection.*

- uint8\_t [outdiv4](#)  
*OUTDIV setting.*

#### 55.13.2.1.0.15 Field Documentation

55.13.2.1.0.15.1 clock\_pllfl\_sel\_t sim\_config\_k21fa12\_t::pllFlSel

55.13.2.1.0.15.2 clock\_er32k\_src\_t sim\_config\_k21fa12\_t::er32kSrc

55.13.2.1.0.15.3 uint8\_t sim\_config\_k21fa12\_t::outdiv4

#### 55.13.3 Macro Definition Documentation

55.13.3.1 #define SDHC\_EXT\_CLK\_COUNT 1

55.13.3.2 #define USB\_EXT\_CLK\_COUNT 1

55.13.3.3 #define FTM\_EXT\_CLK\_COUNT 2

#### 55.13.4 Variable Documentation

55.13.4.1 uint32\_t g\_sdhcClkInFreq[SDHC\_EXT\_CLK\_COUNT]

55.13.4.2 uint32\_t g\_usbClkInFreq[USB\_EXT\_CLK\_COUNT]

55.13.4.3 uint32\_t g\_ftmClkFreq[FTM\_EXT\_CLK\_COUNT]

## K22F12810

### 55.14 K22F12810

The data structure definition for K22F12810 clock manager.

#### 55.14.1 Overview

##### Files

- file [fsl\\_clock\\_MK22F12810.h](#)

##### Data Structures

- 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.*

##### 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.*

#### 55.14.2 Data Structure Documentation

##### 55.14.2.1 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.*



#### 55.14.2.1.0.16 Field Documentation

55.14.2.1.0.16.1 `clock_pllfl_sel_t sim_config_k22f12810_t::pllFlSel`

55.14.2.1.0.16.2 `clock_er32k_src_t sim_config_k22f12810_t::er32kSrc`

55.14.2.1.0.16.3 `uint8_t sim_config_k22f12810_t::outdiv4`

#### 55.14.3 Macro Definition Documentation

55.14.3.1 `#define USB_EXT_CLK_COUNT 1`

55.14.3.2 `#define FTM_EXT_CLK_COUNT 2`

#### 55.14.4 Variable Documentation

55.14.4.1 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

55.14.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K22F25612

### 55.15 K22F25612

The data structure definition for K22F25612 clock manager.

#### 55.15.1 Overview

##### Files

- file [fsl\\_clock\\_MK22F25612.h](#)

##### Data Structures

- 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.*

##### 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.*

#### 55.15.2 Data Structure Documentation

##### 55.15.2.1 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.*

#### 55.15.2.1.0.17 Field Documentation

55.15.2.1.0.17.1 `clock_pllfl_sel_t sim_config_k22f25612_t::pllFlSel`

55.15.2.1.0.17.2 `clock_er32k_src_t sim_config_k22f25612_t::er32kSrc`

55.15.2.1.0.17.3 `uint8_t sim_config_k22f25612_t::outdiv4`

#### 55.15.3 Macro Definition Documentation

55.15.3.1 `#define USB_EXT_CLK_COUNT 1`

55.15.3.2 `#define FTM_EXT_CLK_COUNT 2`

#### 55.15.4 Variable Documentation

55.15.4.1 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

55.15.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K22F51212

### 55.16 K22F51212

The data structure definition for K22F51212 clock manager.

#### 55.16.1 Overview

##### Files

- file [fsl\\_clock\\_MK22F51212.h](#)

##### Data Structures

- 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.*

##### 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.*

#### 55.16.2 Data Structure Documentation

##### 55.16.2.1 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.*

#### 55.16.2.1.0.18 Field Documentation

55.16.2.1.0.18.1 `clock_pllfl_sel_t sim_config_k22f51212_t::pllFlSel`

55.16.2.1.0.18.2 `clock_er32k_src_t sim_config_k22f51212_t::er32kSrc`

55.16.2.1.0.18.3 `uint8_t sim_config_k22f51212_t::outdiv4`

#### 55.16.3 Macro Definition Documentation

55.16.3.1 `#define USB_EXT_CLK_COUNT 1`

55.16.3.2 `#define FTM_EXT_CLK_COUNT 2`

#### 55.16.4 Variable Documentation

55.16.4.1 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

55.16.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K24F12

### 55.17 K24F12

The data structure definition for K24F12 clock manager.

#### 55.17.1 Overview

##### Files

- file [fsl\\_clock\\_MK24F12.h](#)

##### Data Structures

- 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.*

##### 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).*

#### 55.17.2 Data Structure Documentation

##### 55.17.2.1 struct sim\_config\_k24f12\_t

##### Data Fields

- [clock\\_pllfl\\_sel\\_t](#) [pllFlISel](#)  
*PLL/FLL/IRC48M selection.*
- [clock\\_er32k\\_src\\_t](#) [er32kSrc](#)  
*ERCLK32K source selection.*

- uint8\_t [outdiv4](#)  
*OUTDIV setting.*

#### 55.17.2.1.0.19 Field Documentation

55.17.2.1.0.19.1 clock\_pllfl\_sel\_t sim\_config\_k24f12\_t::pllFlSel

55.17.2.1.0.19.2 clock\_er32k\_src\_t sim\_config\_k24f12\_t::er32kSrc

55.17.2.1.0.19.3 uint8\_t sim\_config\_k24f12\_t::outdiv4

#### 55.17.3 Macro Definition Documentation

55.17.3.1 #define SDHC\_EXT\_CLK\_COUNT 1

55.17.3.2 #define USB\_EXT\_CLK\_COUNT 1

55.17.3.3 #define FTM\_EXT\_CLK\_COUNT 2

#### 55.17.4 Variable Documentation

55.17.4.1 uint32\_t g\_sdhcClkInFreq[SDHC\_EXT\_CLK\_COUNT]

55.17.4.2 uint32\_t g\_usbClkInFreq[USB\_EXT\_CLK\_COUNT]

55.17.4.3 uint32\_t g\_ftmClkFreq[FTM\_EXT\_CLK\_COUNT]

## K24F25612

### 55.18 K24F25612

The data structure definition for K24F25612 clock manager.

#### 55.18.1 Overview

##### Files

- file [fsl\\_clock\\_MK24F25612.h](#)

##### Data Structures

- 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.*

##### 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).*

#### 55.18.2 Data Structure Documentation

##### 55.18.2.1 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.*



#### 55.18.2.1.0.20 Field Documentation

55.18.2.1.0.20.1 `clock_pllfl_sel_t sim_config_k24f25612_t::pllFlSel`

55.18.2.1.0.20.2 `clock_er32k_src_t sim_config_k24f25612_t::er32kSrc`

55.18.2.1.0.20.3 `uint8_t sim_config_k24f25612_t::outdiv4`

#### 55.18.3 Macro Definition Documentation

55.18.3.1 `#define USB_EXT_CLK_COUNT 1`

55.18.3.2 `#define FTM_EXT_CLK_COUNT 2`

#### 55.18.4 Variable Documentation

55.18.4.1 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

55.18.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K30D10

### 55.19 K30D10

The data structure definition for K30D10 clock manager.

#### 55.19.1 Overview

##### Files

- file [fsl\\_clock\\_MK30D10.h](#)

##### Data Structures

- struct [sim\\_config\\_k30d10\\_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.*

##### 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\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency([FTM\\_CLK](#)).*

#### 55.19.2 Data Structure Documentation

##### 55.19.2.1 struct sim\_config\_k30d10\_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.*

#### 55.19.2.1.0.21 Field Documentation

55.19.2.1.0.21.1 `clock_pllfl_sel_t sim_config_k30d10_t::pllFlSel`

55.19.2.1.0.21.2 `clock_er32k_src_t sim_config_k30d10_t::er32kSrc`

55.19.2.1.0.21.3 `uint8_t sim_config_k30d10_t::outdiv4`

#### 55.19.3 Macro Definition Documentation

55.19.3.1 `#define ENET_EXT_CLK_COUNT 1`

55.19.3.2 `#define SDHC_EXT_CLK_COUNT 1`

55.19.3.3 `#define USB_EXT_CLK_COUNT 1`

55.19.3.4 `#define FTM_EXT_CLK_COUNT 2`

#### 55.19.4 Variable Documentation

55.19.4.1 `uint32_t g_sdhcClkInFreq[SDHC_EXT_CLK_COUNT]`

55.19.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K40D10

### 55.20 K40D10

The data structure definition for K40D10 clock manager.

#### 55.20.1 Overview

##### Files

- file [fsl\\_clock\\_MK40D10.h](#)

##### Data Structures

- struct [sim\\_config\\_k40d10\\_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.*

##### 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\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency([FTM\\_CLK](#)).*

#### 55.20.2 Data Structure Documentation

##### 55.20.2.1 struct [sim\\_config\\_k40d10\\_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.*

#### 55.20.2.1.0.22 Field Documentation

55.20.2.1.0.22.1 `clock_pllfl_sel_t sim_config_k40d10_t::pllFlSel`

55.20.2.1.0.22.2 `clock_er32k_src_t sim_config_k40d10_t::er32kSrc`

55.20.2.1.0.22.3 `uint8_t sim_config_k40d10_t::outdiv4`

#### 55.20.3 Macro Definition Documentation

55.20.3.1 `#define ENET_EXT_CLK_COUNT 1`

55.20.3.2 `#define SDHC_EXT_CLK_COUNT 1`

55.20.3.3 `#define USB_EXT_CLK_COUNT 1`

55.20.3.4 `#define FTM_EXT_CLK_COUNT 2`

#### 55.20.4 Variable Documentation

55.20.4.1 `uint32_t g_sdhcClkInFreq[SDHC_EXT_CLK_COUNT]`

55.20.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K50D10

### 55.21 K50D10

The data structure definition for K50D10 clock manager.

#### 55.21.1 Overview

##### Files

- file [fsl\\_clock\\_MK50D10.h](#)

##### Data Structures

- struct [sim\\_config\\_k50d10\\_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.*

##### 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\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency([FTM\\_CLK](#)).*

#### 55.21.2 Data Structure Documentation

##### 55.21.2.1 struct [sim\\_config\\_k50d10\\_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.*

#### 55.21.2.1.0.23 Field Documentation

55.21.2.1.0.23.1 `clock_pllfl_sel_t sim_config_k50d10_t::pllFlSel`

55.21.2.1.0.23.2 `clock_er32k_src_t sim_config_k50d10_t::er32kSrc`

55.21.2.1.0.23.3 `uint8_t sim_config_k50d10_t::outdiv4`

#### 55.21.3 Macro Definition Documentation

55.21.3.1 `#define ENET_EXT_CLK_COUNT 1`

55.21.3.2 `#define SDHC_EXT_CLK_COUNT 1`

55.21.3.3 `#define USB_EXT_CLK_COUNT 1`

55.21.3.4 `#define FTM_EXT_CLK_COUNT 2`

#### 55.21.4 Variable Documentation

55.21.4.1 `uint32_t g_sdhcClkInFreq[SDHC_EXT_CLK_COUNT]`

55.21.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K51D10

### 55.22 K51D10

The data structure definition for K51D10 clock manager.

#### 55.22.1 Overview

##### Files

- file [fsl\\_clock\\_MK51D10.h](#)

##### Data Structures

- struct [sim\\_config\\_k51d10\\_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.*

##### 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\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency([FTM\\_CLK](#)).*

#### 55.22.2 Data Structure Documentation

##### 55.22.2.1 struct [sim\\_config\\_k51d10\\_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.*

#### 55.22.2.1.0.24 Field Documentation

55.22.2.1.0.24.1 `clock_pllfl_sel_t sim_config_k51d10_t::pllFlSel`

55.22.2.1.0.24.2 `clock_er32k_src_t sim_config_k51d10_t::er32kSrc`

55.22.2.1.0.24.3 `uint8_t sim_config_k51d10_t::outdiv4`

#### 55.22.3 Macro Definition Documentation

55.22.3.1 `#define ENET_EXT_CLK_COUNT 1`

55.22.3.2 `#define SDHC_EXT_CLK_COUNT 1`

55.22.3.3 `#define USB_EXT_CLK_COUNT 1`

55.22.3.4 `#define FTM_EXT_CLK_COUNT 2`

#### 55.22.4 Variable Documentation

55.22.4.1 `uint32_t g_sdhcClkInFreq[SDHC_EXT_CLK_COUNT]`

55.22.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K52D10

### 55.23 K52D10

The data structure definition for K52D10 clock manager.

#### 55.23.1 Overview

##### Files

- file [fsl\\_clock\\_MK52D10.h](#)

##### Data Structures

- struct [sim\\_config\\_k52d10\\_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.*

##### 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\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency([FTM\\_CLK](#)).*

#### 55.23.2 Data Structure Documentation

##### 55.23.2.1 struct [sim\\_config\\_k52d10\\_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.*

#### 55.23.2.1.0.25 Field Documentation

55.23.2.1.0.25.1 `clock_pllfl_sel_t sim_config_k52d10_t::pllFlSel`

55.23.2.1.0.25.2 `clock_er32k_src_t sim_config_k52d10_t::er32kSrc`

55.23.2.1.0.25.3 `uint8_t sim_config_k52d10_t::outdiv4`

#### 55.23.3 Macro Definition Documentation

55.23.3.1 `#define ENET_EXT_CLK_COUNT 1`

55.23.3.2 `#define SDHC_EXT_CLK_COUNT 1`

55.23.3.3 `#define USB_EXT_CLK_COUNT 1`

55.23.3.4 `#define FTM_EXT_CLK_COUNT 2`

#### 55.23.4 Variable Documentation

55.23.4.1 `uint32_t g_sdhcClkInFreq[SDHC_EXT_CLK_COUNT]`

55.23.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K53D10

### 55.24 K53D10

The data structure definition for K53D10 clock manager.

#### 55.24.1 Overview

##### Files

- file [fsl\\_clock\\_MK53D10.h](#)

##### Data Structures

- struct [sim\\_config\\_k53d10\\_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.*

##### 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\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency([FTM\\_CLK](#)).*

#### 55.24.2 Data Structure Documentation

##### 55.24.2.1 struct [sim\\_config\\_k53d10\\_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.*

#### 55.24.2.1.0.26 Field Documentation

55.24.2.1.0.26.1 `clock_pllfl_sel_t sim_config_k53d10_t::pllFlSel`

55.24.2.1.0.26.2 `clock_er32k_src_t sim_config_k53d10_t::er32kSrc`

55.24.2.1.0.26.3 `uint8_t sim_config_k53d10_t::outdiv4`

#### 55.24.3 Macro Definition Documentation

55.24.3.1 `#define ENET_EXT_CLK_COUNT 1`

55.24.3.2 `#define SDHC_EXT_CLK_COUNT 1`

55.24.3.3 `#define USB_EXT_CLK_COUNT 1`

55.24.3.4 `#define FTM_EXT_CLK_COUNT 2`

#### 55.24.4 Variable Documentation

55.24.4.1 `uint32_t g_sdhcClkInFreq[SDHC_EXT_CLK_COUNT]`

55.24.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K60D10

### 55.25 K60D10

The data structure definition for K60D10 clock manager.

#### 55.25.1 Overview

K60D10 clock manager code is shared by K10D10, K20D10, K30D10, K40D10, K50D10, K51D10, K52D10, K53D10 and K60D10.

#### Files

- file [fsl\\_clock\\_MK60D10.h](#)

#### Data Structures

- 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.*

#### 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\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency(FTM\_CLK).*

## 55.25.2 Data Structure Documentation

### 55.25.2.1 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.*

#### 55.25.2.1.0.27 Field Documentation

55.25.2.1.0.27.1 `clock_pllfl_sel_t sim_config_k60d10_t::pllFlSel`

55.25.2.1.0.27.2 `clock_er32k_src_t sim_config_k60d10_t::er32kSrc`

55.25.2.1.0.27.3 `uint8_t sim_config_k60d10_t::outdiv4`

## 55.25.3 Macro Definition Documentation

55.25.3.1 `#define ENET_EXT_CLK_COUNT 1`

55.25.3.2 `#define SDHC_EXT_CLK_COUNT 1`

55.25.3.3 `#define USB_EXT_CLK_COUNT 1`

55.25.3.4 `#define FTM_EXT_CLK_COUNT 2`

## 55.25.4 Variable Documentation

55.25.4.1 `uint32_t g_sdhcClkInFreq[SDHC_EXT_CLK_COUNT]`

55.25.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K63F12

### 55.26 K63F12

The data structure definition for K63F12 clock manager.

#### 55.26.1 Overview

##### Files

- file [fsl\\_clock\\_MK63F12.h](#)

##### Data Structures

- struct [sim\\_config\\_k63f12\\_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.*

##### 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).*

#### 55.26.2 Data Structure Documentation

##### 55.26.2.1 struct sim\_config\_k63f12\_t

##### Data Fields

- [clock\\_pllfl\\_sel\\_t](#) [pllFlISel](#)



- *PLL/FLL/IRC48M selection.*
  - `clock_er32k_src_t er32kSrc`
    - *ERCLK32K source selection.*
  - `uint8_t outdiv4`
    - *OUTDIV setting.*

#### 55.26.2.1.0.28 Field Documentation

55.26.2.1.0.28.1 `clock_pllfl_sel_t sim_config_k63f12_t::pllFlSel`

55.26.2.1.0.28.2 `clock_er32k_src_t sim_config_k63f12_t::er32kSrc`

55.26.2.1.0.28.3 `uint8_t sim_config_k63f12_t::outdiv4`

#### 55.26.3 Macro Definition Documentation

55.26.3.1 `#define ENET_EXT_CLK_COUNT 1`

55.26.3.2 `#define SDHC_EXT_CLK_COUNT 1`

55.26.3.3 `#define USB_EXT_CLK_COUNT 1`

55.26.3.4 `#define FTM_EXT_CLK_COUNT 2`

#### 55.26.4 Variable Documentation

55.26.4.1 `uint32_t g_sdhcClkInFreq[SDHC_EXT_CLK_COUNT]`

55.26.4.2 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

55.26.4.3 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## K64F12

### 55.27 K64F12

The data structure definition for K64F12 clock manager.

#### 55.27.1 Overview

#### Files

- file [fsl\\_clock\\_MK26F18.h](#)
- file [fsl\\_clock\\_MK64F12.h](#)
- file [fsl\\_clock\\_MK65F18.h](#)
- file [fsl\\_clock\\_MK66F18.h](#)

#### Data Structures

- struct [sim\\_config\\_k26f18\\_t](#)  
*SIM configuration structure for dynamic clock setting. [More...](#)*
- struct [sim\\_config\\_k64f12\\_t](#)  
*SIM configuration structure for dynamic clock setting. [More...](#)*
- struct [sim\\_config\\_k65f18\\_t](#)  
*SIM configuration structure for dynamic clock setting. [More...](#)*
- struct [sim\\_config\\_k66f18\\_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 [TPM\\_EXT\\_CLK\\_COUNT](#) 2  
*TPM external clock source count.*
- #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 [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 **TPM\_EXT\_CLK\_COUNT** 2
- *TPM external clock source count.*
- #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 **TPM\_EXT\_CLK\_COUNT** 2
- *TPM external clock source count.*

## 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\_fmClkFreq** [**FTM\_EXT\_CLK\_COUNT**]  
*FTM external clock frequency(FTM\_CLK).*
- uint32\_t **g\_tpmClkFreq** [**TPM\_EXT\_CLK\_COUNT**]  
*TPM external clock frequency(TPM\_CLK).*
- 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\_fmClkFreq** [**FTM\_EXT\_CLK\_COUNT**]  
*FTM external clock frequency(FTM\_CLK).*
- 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\_fmClkFreq** [**FTM\_EXT\_CLK\_COUNT**]  
*FTM external clock frequency(FTM\_CLK).*
- uint32\_t **g\_tpmClkFreq** [**TPM\_EXT\_CLK\_COUNT**]  
*TPM external clock frequency(TPM\_CLK).*
- 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).*

## K64F12

- uint32\_t [g\\_ftmClkFreq](#) [FTM\_EXT\_CLK\_COUNT]  
*FTM external clock frequency(FTM\_CLK).*
- uint32\_t [g\\_tpmClkFreq](#) [TPM\_EXT\_CLK\_COUNT]  
*TPM external clock frequency(TPM\_CLK).*

## 55.27.2 Data Structure Documentation

### 55.27.2.1 struct sim\_config\_k26f18\_t

#### Data Fields

- [clock\\_pllfl\\_sel\\_t](#) [pllFlSel](#)  
*PLL/FLL/IRC48M selection.*
- uint8\_t [pllflfrac](#)  
*PLL/FLL fractional divider setting.*
- [clock\\_er32k\\_src\\_t](#) [er32kSrc](#)  
*ERCLK32K source selection.*
- uint8\_t [outdiv4](#)  
*OUTDIV setting.*

#### 55.27.2.1.0.29 Field Documentation

55.27.2.1.0.29.1 [clock\\_pllfl\\_sel\\_t](#) [sim\\_config\\_k26f18\\_t::pllFlSel](#)

55.27.2.1.0.29.2 [clock\\_er32k\\_src\\_t](#) [sim\\_config\\_k26f18\\_t::er32kSrc](#)

55.27.2.1.0.29.3 [uint8\\_t](#) [sim\\_config\\_k26f18\\_t::outdiv4](#)

### 55.27.2.2 struct sim\_config\_k64f12\_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.*

**55.27.2.2.0.30 Field Documentation**
**55.27.2.2.0.30.1** `clock_pllfl_sel_t sim_config_k64f12_t::pllFlSel`
**55.27.2.2.0.30.2** `clock_er32k_src_t sim_config_k64f12_t::er32kSrc`
**55.27.2.2.0.30.3** `uint8_t sim_config_k64f12_t::outdiv4`
**55.27.2.3 struct sim\_config\_k65f18\_t**
**Data Fields**

- [clock\\_pllfl\\_sel\\_t pllFlSel](#)  
*PLL/FLL/IRC48M selection.*
- `uint8_t pllflfrac`  
*PLL/FLL fractional divider setting.*
- [clock\\_er32k\\_src\\_t er32kSrc](#)  
*ERCLK32K source selection.*
- `uint8_t outdiv4`  
*OUTDIV setting.*

**55.27.2.3.0.31 Field Documentation**
**55.27.2.3.0.31.1** `clock_pllfl_sel_t sim_config_k65f18_t::pllFlSel`
**55.27.2.3.0.31.2** `clock_er32k_src_t sim_config_k65f18_t::er32kSrc`
**55.27.2.3.0.31.3** `uint8_t sim_config_k65f18_t::outdiv4`
**55.27.2.4 struct sim\_config\_k66f18\_t**
**Data Fields**

- [clock\\_pllfl\\_sel\\_t pllFlSel](#)  
*PLL/FLL/IRC48M selection.*
- `uint8_t pllflfrac`  
*PLL/FLL fractional divider setting.*
- [clock\\_er32k\\_src\\_t er32kSrc](#)  
*ERCLK32K source selection.*
- `uint8_t outdiv4`  
*OUTDIV setting.*



**55.27.2.4.0.32 Field Documentation**

55.27.2.4.0.32.1 `clock_pllfl_sel_t sim_config_k66f18_t::pllFlSel`

55.27.2.4.0.32.2 `clock_er32k_src_t sim_config_k66f18_t::er32kSrc`

55.27.2.4.0.32.3 `uint8_t sim_config_k66f18_t::outdiv4`

**55.27.3 Macro Definition Documentation**

55.27.3.1 `#define SDHC_EXT_CLK_COUNT 1`

55.27.3.2 `#define USB_EXT_CLK_COUNT 1`

55.27.3.3 `#define FTM_EXT_CLK_COUNT 2`

55.27.3.4 `#define TPM_EXT_CLK_COUNT 2`

55.27.3.5 `#define ENET_EXT_CLK_COUNT 1`

55.27.3.6 `#define SDHC_EXT_CLK_COUNT 1`

55.27.3.7 `#define USB_EXT_CLK_COUNT 1`

55.27.3.8 `#define FTM_EXT_CLK_COUNT 2`

55.27.3.9 `#define ENET_EXT_CLK_COUNT 1`

55.27.3.10 `#define SDHC_EXT_CLK_COUNT 1`

55.27.3.11 `#define USB_EXT_CLK_COUNT 1`

55.27.3.12 `#define FTM_EXT_CLK_COUNT 2`

55.27.3.13 `#define TPM_EXT_CLK_COUNT 2`

55.27.3.14 `#define ENET_EXT_CLK_COUNT 1`

55.27.3.15 `#define SDHC_EXT_CLK_COUNT 1`

55.27.3.16 `#define USB_EXT_CLK_COUNT 1`

55.27.3.17 `#define FTM_EXT_CLK_COUNT 2`

55.27.3.18 `#define TPM_EXT_CLK_COUNT 2`

**55.27.4 Variable Documentation**

## KL02Z4

### 55.28 KL02Z4

The data structure definition for KL02Z4 clock manager.

#### 55.28.1 Overview

##### Files

- file [fsl\\_clock\\_MKL02Z4.h](#)

##### Data Structures

- struct [sim\\_config\\_kl02z4\\_t](#)  
*SIM configuration structure for dynamic clock setting. [More...](#)*

##### Macros

- #define [TPM\\_EXT\\_CLK\\_COUNT](#) 2  
*TPM external clock source count.*

##### Variables

- uint32\_t [g\\_tpmClkFreq](#) [[TPM\\_EXT\\_CLK\\_COUNT](#)]  
*TPM external clock frequency(TPM\_CLK).*

#### 55.28.2 Data Structure Documentation

##### 55.28.2.1 struct sim\_config\_kl02z4\_t

##### Data Fields

- uint8\_t [outdiv4](#)  
*OUTDIV setting.*



### 55.28.2.1.0.33 Field Documentation

55.28.2.1.0.33.1 `uint8_t sim_config_kl02z4_t::outdiv4`

### 55.28.3 Macro Definition Documentation

55.28.3.1 `#define TPM_EXT_CLK_COUNT 2`

### 55.28.4 Variable Documentation

55.28.4.1 `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

## KL03Z4

### 55.29 KL03Z4

The data structure definition for KL03Z4 clock manager.

#### 55.29.1 Overview

##### Files

- file [fsl\\_clock\\_MKL03Z4.h](#)

##### Data Structures

- 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.*

##### Variables

- uint32\_t [g\\_tpmClkFreq](#) [[TPM\\_EXT\\_CLK\\_COUNT](#)]  
*TPM external clock frequency(TPM\_CLK).*

#### 55.29.2 Data Structure Documentation

##### 55.29.2.1 struct [sim\\_config\\_kl03z4\\_t](#)

##### Data Fields

- [clock\\_er32k\\_src\\_t](#) [er32kSrc](#)  
*ERCLK32K source selection.*
- uint8\_t [outdiv4](#)  
*OUTDIV setting.*

#### 55.29.2.1.0.34 Field Documentation

55.29.2.1.0.34.1 `clock_er32k_src_t sim_config_kl03z4_t::er32kSrc`

55.29.2.1.0.34.2 `uint8_t sim_config_kl03z4_t::outdiv4`

#### 55.29.3 Macro Definition Documentation

55.29.3.1 `#define TPM_EXT_CLK_COUNT 2`

#### 55.29.4 Variable Documentation

55.29.4.1 `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

---

## KL13Z644

### 55.30 KL13Z644

The data structure definition for KL13Z644 clock manager.

## 55.31 KL14Z4

The data structure definition for KL14Z4 clock manager.

### 55.31.1 Overview

#### Files

- file [fsl\\_clock\\_MKL14Z4.h](#)

#### Data Structures

- struct [sim\\_config\\_kl14z4\\_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.*

#### 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).*

### 55.31.2 Data Structure Documentation

#### 55.31.2.1 struct sim\_config\_kl14z4\_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.*

## **KL14Z4**

### **55.31.2.1.0.35 Field Documentation**

**55.31.2.1.0.35.1** `clock_pllfl_sel_t sim_config_kl14z4_t::pllFlSel`

**55.31.2.1.0.35.2** `clock_er32k_src_t sim_config_kl14z4_t::er32kSrc`

**55.31.2.1.0.35.3** `uint8_t sim_config_kl14z4_t::outdiv4`

### **55.31.3 Macro Definition Documentation**

**55.31.3.1** `#define TPM_EXT_CLK_COUNT 2`

**55.31.3.2** `#define USB_EXT_CLK_COUNT 1`

### **55.31.4 Variable Documentation**

**55.31.4.1** `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

**55.31.4.2** `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

## 55.32 KL15Z4

The data structure definition for KL15Z4 clock manager.

### 55.32.1 Overview

#### Files

- file [fsl\\_clock\\_MKL15Z4.h](#)

#### Data Structures

- struct [sim\\_config\\_kl15z4\\_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.*

#### 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).*

### 55.32.2 Data Structure Documentation

#### 55.32.2.1 struct sim\_config\_kl15z4\_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.*

## **KL15Z4**

### **55.32.2.1.0.36 Field Documentation**

**55.32.2.1.0.36.1** `clock_pllfl_sel_t sim_config_kl15z4_t::pllFlSel`

**55.32.2.1.0.36.2** `clock_er32k_src_t sim_config_kl15z4_t::er32kSrc`

**55.32.2.1.0.36.3** `uint8_t sim_config_kl15z4_t::outdiv4`

### **55.32.3 Macro Definition Documentation**

**55.32.3.1** `#define TPM_EXT_CLK_COUNT 2`

**55.32.3.2** `#define USB_EXT_CLK_COUNT 1`

### **55.32.4 Variable Documentation**

**55.32.4.1** `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

**55.32.4.2** `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`



## 55.33 KL16Z4

The data structure definition for KL16Z4 clock manager.

### 55.33.1 Overview

#### Files

- file [fsl\\_clock\\_MKL16Z4.h](#)

#### Data Structures

- struct [sim\\_config\\_kl16z4\\_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.*

#### Variables

- uint32\_t [g\\_tpmClkFreq](#) [[TPM\\_EXT\\_CLK\\_COUNT](#)]  
*TPM external clock frequency(TPM\_CLK).*

### 55.33.2 Data Structure Documentation

#### 55.33.2.1 struct sim\_config\_kl16z4\_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.*

## **KL16Z4**

### **55.33.2.1.0.37 Field Documentation**

**55.33.2.1.0.37.1** `clock_pllfl_sel_t sim_config_kl16z4_t::pllFlSel`

**55.33.2.1.0.37.2** `clock_er32k_src_t sim_config_kl16z4_t::er32kSrc`

**55.33.2.1.0.37.3** `uint8_t sim_config_kl16z4_t::outdiv4`

### **55.33.3 Macro Definition Documentation**

**55.33.3.1** `#define TPM_EXT_CLK_COUNT 2`

**55.33.3.2** `#define USB_EXT_CLK_COUNT 1`

### **55.33.4 Variable Documentation**

**55.33.4.1** `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

## 55.34 KL17Z4

The data structure definition for KL17Z4 clock manager.

### 55.34.1 Overview

#### Files

- file [fsl\\_clock\\_MKL17Z4.h](#)

#### Data Structures

- struct [sim\\_config\\_kl17z4\\_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.*

#### Variables

- uint32\_t [g\\_tpmClkFreq](#) [[TPM\\_EXT\\_CLK\\_COUNT](#)]  
*TPM external clock frequency(TPM\_CLK).*

### 55.34.2 Data Structure Documentation

#### 55.34.2.1 struct sim\_config\_kl17z4\_t

##### Data Fields

- [clock\\_er32k\\_src\\_t](#) [er32kSrc](#)  
*ERCLK32K source selection.*
- uint8\_t [outdiv4](#)  
*OUTDIV setting.*

## **KL17Z4**

### **55.34.2.1.0.38 Field Documentation**

**55.34.2.1.0.38.1** `clock_er32k_src_t sim_config_kl17z4_t::er32kSrc`

**55.34.2.1.0.38.2** `uint8_t sim_config_kl17z4_t::outdiv4`

### **55.34.3 Macro Definition Documentation**

**55.34.3.1** `#define TPM_EXT_CLK_COUNT 2`

**55.34.3.2** `#define USB_EXT_CLK_COUNT 1`

### **55.34.4 Variable Documentation**

**55.34.4.1** `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

## 55.35 KL17Z644

The data structure definition for KL17Z644 clock manager.

### 55.35.1 Overview

#### Files

- file [fsl\\_clock\\_MKL17Z644.h](#)

#### Data Structures

- struct [sim\\_config\\_kl17z644\\_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.*

#### 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).*

### 55.35.2 Data Structure Documentation

#### 55.35.2.1 struct sim\_config\_kl17z644\_t

##### Data Fields

- [clock\\_er32k\\_src\\_t](#) [er32kSrc](#)  
*ERCLK32K source selection.*
- uint8\_t [outdiv4](#)  
*OUTDIV setting.*

## **KL17Z644**

### **55.35.2.1.0.39 Field Documentation**

**55.35.2.1.0.39.1** `clock_er32k_src_t sim_config_kl17z644_t::er32kSrc`

**55.35.2.1.0.39.2** `uint8_t sim_config_kl17z644_t::outdiv4`

### **55.35.3 Macro Definition Documentation**

**55.35.3.1** `#define TPM_EXT_CLK_COUNT 2`

**55.35.3.2** `#define USB_EXT_CLK_COUNT 1`

### **55.35.4 Variable Documentation**

**55.35.4.1** `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

**55.35.4.2** `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

## 55.36 KL24Z4

The data structure definition for KL24Z4 clock manager.

### 55.36.1 Overview

#### Files

- file [fsl\\_clock\\_MKL24Z4.h](#)

#### Data Structures

- struct [sim\\_config\\_kl24z4\\_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.*

#### 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).*

### 55.36.2 Data Structure Documentation

#### 55.36.2.1 struct sim\_config\_kl24z4\_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.*

## **KL24Z4**

### **55.36.2.1.0.40 Field Documentation**

**55.36.2.1.0.40.1** `clock_pllfl_sel_t sim_config_kl24z4_t::pllFlSel`

**55.36.2.1.0.40.2** `clock_er32k_src_t sim_config_kl24z4_t::er32kSrc`

**55.36.2.1.0.40.3** `uint8_t sim_config_kl24z4_t::outdiv4`

### **55.36.3 Macro Definition Documentation**

**55.36.3.1** `#define TPM_EXT_CLK_COUNT 2`

**55.36.3.2** `#define USB_EXT_CLK_COUNT 1`

### **55.36.4 Variable Documentation**

**55.36.4.1** `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

**55.36.4.2** `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`



## 55.37 KL25Z4

The data structure definition for KL25Z4 clock manager.

### 55.37.1 Overview

#### Files

- file [fsl\\_clock\\_MKL25Z4.h](#)

#### Data Structures

- struct [sim\\_config\\_kl25z4\\_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.*

#### 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).*

### 55.37.2 Data Structure Documentation

#### 55.37.2.1 struct sim\_config\_kl25z4\_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.*

## **KL25Z4**

### **55.37.2.1.0.41 Field Documentation**

**55.37.2.1.0.41.1** `clock_pllfl_sel_t sim_config_kl25z4_t::pllFlSel`

**55.37.2.1.0.41.2** `clock_er32k_src_t sim_config_kl25z4_t::er32kSrc`

**55.37.2.1.0.41.3** `uint8_t sim_config_kl25z4_t::outdiv4`

### **55.37.3 Macro Definition Documentation**

**55.37.3.1** `#define TPM_EXT_CLK_COUNT 2`

**55.37.3.2** `#define USB_EXT_CLK_COUNT 1`

### **55.37.4 Variable Documentation**

**55.37.4.1** `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

**55.37.4.2** `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

## 55.38 KL26Z4

The data structure definition for KL26Z4 clock manager.

### 55.38.1 Overview

#### Files

- file [fsl\\_clock\\_MKL26Z4.h](#)

#### Data Structures

- struct [sim\\_config\\_kl26z4\\_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.*

#### Variables

- uint32\_t [g\\_tpmClkFreq](#) [[TPM\\_EXT\\_CLK\\_COUNT](#)]  
*TPM external clock frequency(TPM\_CLK).*

### 55.38.2 Data Structure Documentation

#### 55.38.2.1 struct [sim\\_config\\_kl26z4\\_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.*

## **KL26Z4**

### **55.38.2.1.0.42 Field Documentation**

**55.38.2.1.0.42.1** `clock_pllfl_sel_t sim_config_kl26z4_t::pllFlSel`

**55.38.2.1.0.42.2** `clock_er32k_src_t sim_config_kl26z4_t::er32kSrc`

**55.38.2.1.0.42.3** `uint8_t sim_config_kl26z4_t::outdiv4`

### **55.38.3 Macro Definition Documentation**

**55.38.3.1** `#define TPM_EXT_CLK_COUNT 2`

**55.38.3.2** `#define USB_EXT_CLK_COUNT 1`

### **55.38.4 Variable Documentation**

**55.38.4.1** `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

## 55.39 KL27Z4

The data structure definition for KL27Z4 clock manager.

### 55.39.1 Overview

#### Files

- file [fsl\\_clock\\_MKL27Z4.h](#)

#### Data Structures

- struct [sim\\_config\\_kl27z4\\_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.*

#### Variables

- uint32\_t [g\\_tpmClkFreq](#) [[TPM\\_EXT\\_CLK\\_COUNT](#)]  
*TPM external clock frequency(TPM\_CLK).*

### 55.39.2 Data Structure Documentation

#### 55.39.2.1 struct [sim\\_config\\_kl27z4\\_t](#)

##### Data Fields

- [clock\\_er32k\\_src\\_t](#) [er32kSrc](#)  
*ERCLK32K source selection.*
- uint8\_t [outdiv4](#)  
*OUTDIV setting.*

## **KL27Z4**

### **55.39.2.1.0.43 Field Documentation**

**55.39.2.1.0.43.1** `clock_er32k_src_t sim_config_kl27z4_t::er32kSrc`

**55.39.2.1.0.43.2** `uint8_t sim_config_kl27z4_t::outdiv4`

### **55.39.3 Macro Definition Documentation**

**55.39.3.1** `#define TPM_EXT_CLK_COUNT 2`

**55.39.3.2** `#define USB_EXT_CLK_COUNT 1`

### **55.39.4 Variable Documentation**

**55.39.4.1** `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

## 55.40 KL127Z644

The data structure definition for KL127Z644 clock manager.

### 55.40.1 Overview

#### Files

- file [fsl\\_clock\\_MKL27Z644.h](#)

#### Data Structures

- struct [sim\\_config\\_kl27z644\\_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.*

#### 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).*

### 55.40.2 Data Structure Documentation

#### 55.40.2.1 struct sim\_config\_kl27z644\_t

##### Data Fields

- [clock\\_er32k\\_src\\_t](#) [er32kSrc](#)  
*ERCLK32K source selection.*
- uint8\_t [outdiv4](#)  
*OUTDIV setting.*

## **KL127Z644**

### **55.40.2.1.0.44 Field Documentation**

**55.40.2.1.0.44.1 clock\_er32k\_src\_t sim\_config\_kl27z644\_t::er32kSrc**

**55.40.2.1.0.44.2 uint8\_t sim\_config\_kl27z644\_t::outdiv4**

### **55.40.3 Macro Definition Documentation**

**55.40.3.1 #define TPM\_EXT\_CLK\_COUNT 2**

**55.40.3.2 #define USB\_EXT\_CLK\_COUNT 1**

### **55.40.4 Variable Documentation**

**55.40.4.1 uint32\_t g\_tpmClkFreq[TPM\_EXT\_CLK\_COUNT]**

**55.40.4.2 uint32\_t g\_usbClkInFreq[USB\_EXT\_CLK\_COUNT]**



## 55.41 K128T7

The data structure definition for K128T7 clock manager.

### 55.41.1 Overview

#### Files

- file [fsl\\_clock\\_MKL28T7.h](#)

#### Data Structures

- struct [sim\\_config\\_kl28t7\\_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](#)).*

### 55.41.2 Data Structure Documentation

#### 55.41.2.1 struct [sim\\_config\\_kl28t7\\_t](#)

##### Data Fields

- [clock\\_er32k\\_src\\_t](#) [er32kSrc](#)  
*ERCLK32K source selection.*

## **K128T7**

### **55.41.2.1.0.45 Field Documentation**

**55.41.2.1.0.45.1 clock\_er32k\_src\_t sim\_config\_kl28t7\_t::er32kSrc**

### **55.41.3 Macro Definition Documentation**

**55.41.3.1 #define TPM\_EXT\_CLK\_COUNT 2**

**55.41.3.2 #define CLOCK\_CONFIG\_NUM 2**

**55.41.3.3 #define CLOCK\_CONFIG\_INDEX\_FOR\_VLPR 0**

**55.41.3.4 #define CLOCK\_CONFIG\_INDEX\_FOR\_RUN 1**

### **55.41.4 Variable Documentation**

**55.41.4.1 uint32\_t g\_tpmClkFreq[TPM\_EXT\_CLK\_COUNT]**

## 55.42 KL33Z4

The data structure definition for KL33Z4 clock manager.

### 55.42.1 Overview

#### Files

- file [fsl\\_clock\\_MKL33Z4.h](#)

#### Data Structures

- struct [sim\\_config\\_kl33z4\\_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.*

#### Variables

- uint32\_t [g\\_tpmClkFreq](#) [[TPM\\_EXT\\_CLK\\_COUNT](#)]  
*TPM external clock frequency(TPM\_CLK).*

### 55.42.2 Data Structure Documentation

#### 55.42.2.1 struct [sim\\_config\\_kl33z4\\_t](#)

##### Data Fields

- [clock\\_er32k\\_src\\_t](#) [er32kSrc](#)  
*ERCLK32K source selection.*
- uint8\_t [outdiv4](#)  
*OUTDIV setting.*

## **KL33Z4**

### **55.42.2.1.0.46 Field Documentation**

**55.42.2.1.0.46.1** `clock_er32k_src_t sim_config_kl33z4_t::er32kSrc`

**55.42.2.1.0.46.2** `uint8_t sim_config_kl33z4_t::outdiv4`

### **55.42.3 Macro Definition Documentation**

**55.42.3.1** `#define TPM_EXT_CLK_COUNT 2`

**55.42.3.2** `#define USB_EXT_CLK_COUNT 1`

### **55.42.4 Variable Documentation**

**55.42.4.1** `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

**55.43 KL33Z644**

The data structure definition for KL33Z644 clock manager.

## KL34Z4

### 55.44 KL34Z4

The data structure definition for KL34Z4 clock manager.

#### 55.44.1 Overview

##### Files

- file [fsl\\_clock\\_MKL34Z4.h](#)

##### Data Structures

- struct [sim\\_config\\_kl34z4\\_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.*

##### Variables

- uint32\_t [g\\_tpmClkFreq](#) [[TPM\\_EXT\\_CLK\\_COUNT](#)]  
*TPM external clock frequency(TPM\_CLK).*

#### 55.44.2 Data Structure Documentation

##### 55.44.2.1 struct sim\_config\_kl34z4\_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.*

#### 55.44.2.1.0.47 Field Documentation

55.44.2.1.0.47.1 `clock_pllfl_sel_t sim_config_kl34z4_t::pllFlSel`

55.44.2.1.0.47.2 `clock_er32k_src_t sim_config_kl34z4_t::er32kSrc`

55.44.2.1.0.47.3 `uint8_t sim_config_kl34z4_t::outdiv4`

#### 55.44.3 Macro Definition Documentation

55.44.3.1 `#define TPM_EXT_CLK_COUNT 2`

55.44.3.2 `#define USB_EXT_CLK_COUNT 1`

#### 55.44.4 Variable Documentation

55.44.4.1 `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

## KL36Z4

### 55.45 KL36Z4

The data structure definition for KL36Z4 clock manager.

#### 55.45.1 Overview

##### Files

- file [fsl\\_clock\\_MKL36Z4.h](#)

##### Data Structures

- struct [sim\\_config\\_kl36z4\\_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.*

##### Variables

- uint32\_t [g\\_tpmClkFreq](#) [[TPM\\_EXT\\_CLK\\_COUNT](#)]  
*TPM external clock frequency(TPM\_CLK).*

#### 55.45.2 Data Structure Documentation

##### 55.45.2.1 struct sim\_config\_kl36z4\_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.*



#### 55.45.2.1.0.48 Field Documentation

55.45.2.1.0.48.1 `clock_pllfl_sel_t sim_config_kl36z4_t::pllFlSel`

55.45.2.1.0.48.2 `clock_er32k_src_t sim_config_kl36z4_t::er32kSrc`

55.45.2.1.0.48.3 `uint8_t sim_config_kl36z4_t::outdiv4`

#### 55.45.3 Macro Definition Documentation

55.45.3.1 `#define TPM_EXT_CLK_COUNT 2`

55.45.3.2 `#define USB_EXT_CLK_COUNT 1`

#### 55.45.4 Variable Documentation

55.45.4.1 `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

## KL43Z4

### 55.46 KL43Z4

The data structure definition for KL43Z4 clock manager.

#### 55.46.1 Overview

KL43Z4 clock manager code is shared by KL17Z4, KL27Z4, KL33Z4 and KL43Z4.

#### Files

- file [fsl\\_clock\\_MKL43Z4.h](#)

#### Data Structures

- struct [sim\\_config\\_kl43z4\\_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.*

#### Variables

- [uint32\\_t g\\_tpmClkFreq \[TPM\\_EXT\\_CLK\\_COUNT\]](#)  
*TPM external clock frequency(TPM\_CLK).*

### 55.46.2 Data Structure Documentation

#### 55.46.2.1 struct sim\_config\_kl43z4\_t

##### Data Fields

- [clock\\_er32k\\_src\\_t er32kSrc](#)  
*ERCLK32K source selection.*
- [uint8\\_t outdiv4](#)  
*OUTDIV setting.*

#### 55.46.2.1.0.49 Field Documentation

55.46.2.1.0.49.1 `clock_er32k_src_t sim_config_kl43z4_t::er32kSrc`

55.46.2.1.0.49.2 `uint8_t sim_config_kl43z4_t::outdiv4`

#### 55.46.3 Macro Definition Documentation

55.46.3.1 `#define TPM_EXT_CLK_COUNT 2`

55.46.3.2 `#define USB_EXT_CLK_COUNT 1`

#### 55.46.4 Variable Documentation

55.46.4.1 `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

## KL46Z4

### 55.47 KL46Z4

The data structure definition for KL46Z4 clock manager.

#### 55.47.1 Overview

The data structure definition for KW01Z4 clock manager.

KL46Z4 clock manager code is shared by KL16Z4, KL26Z4, KL36Z4 and KL46Z4.

#### Files

- file [fsl\\_clock\\_MKL46Z4.h](#)
- file [fsl\\_clock\\_MKW01Z4.h](#)

#### Data Structures

- struct [sim\\_config\\_kl46z4\\_t](#)  
*SIM configuration structure for dynamic clock setting. [More...](#)*
- struct [osc\\_user\\_config\\_kw01z4\\_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 [TPM\\_EXT\\_CLK\\_COUNT](#) 2  
*TPM external clock source count.*
- #define [USB\\_EXT\\_CLK\\_COUNT](#) 1  
*USB external clock source count.*

#### Variables

- uint32\_t [g\\_tpmClkFreq](#) [[TPM\\_EXT\\_CLK\\_COUNT](#)]  
*TPM external clock frequency(TPM\_CLK).*
- uint32\_t [g\\_tpmClkFreq](#) [[TPM\\_EXT\\_CLK\\_COUNT](#)]  
*TPM external clock frequency(TPM\_CLK).*

## 55.47.2 Data Structure Documentation

### 55.47.2.1 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.*

#### 55.47.2.1.0.50 Field Documentation

55.47.2.1.0.50.1 [clock\\_pllfl\\_sel\\_t sim\\_config\\_kl46z4\\_t::pllFlSel](#)

55.47.2.1.0.50.2 [clock\\_er32k\\_src\\_t sim\\_config\\_kl46z4\\_t::er32kSrc](#)

55.47.2.1.0.50.3 [uint8\\_t sim\\_config\\_kl46z4\\_t::outdiv4](#)

### 55.47.2.2 struct osc\_user\_config\_kw01z4\_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.*

## **KL46Z4**

### **55.47.2.2.0.51 Field Documentation**

**55.47.2.2.0.51.1** `clock_pllfl_sel_t osc_user_config_kw01z4_t::pllFlSel`

**55.47.2.2.0.51.2** `clock_er32k_src_t osc_user_config_kw01z4_t::er32kSrc`

**55.47.2.2.0.51.3** `uint8_t osc_user_config_kw01z4_t::outdiv4`

### **55.47.3 Macro Definition Documentation**

**55.47.3.1** `#define TPM_EXT_CLK_COUNT 2`

**55.47.3.2** `#define USB_EXT_CLK_COUNT 1`

**55.47.3.3** `#define TPM_EXT_CLK_COUNT 2`

**55.47.3.4** `#define USB_EXT_CLK_COUNT 1`

### **55.47.4 Variable Documentation**

**55.47.4.1** `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

**55.47.4.2** `uint32_t g_tpmClkFreq[TPM_EXT_CLK_COUNT]`

## 55.48 KV10Z7

The data structure definition for KV10Z7 clock manager.

### 55.48.1 Overview

#### Files

- file [fsl\\_clock\\_MKV10Z7.h](#)

#### Data Structures

- 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.*

#### Variables

- uint32\_t [g\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency(FTM\_CLK).*

### 55.48.2 Data Structure Documentation

#### 55.48.2.1 struct [sim\\_config\\_kv10z7\\_t](#)

##### Data Fields

- [clock\\_er32k\\_src\\_t](#) [er32kSrc](#)  
*ERCLK32K source selection.*
- uint8\_t [outdiv5](#)  
*OUTDIV setting.*
- bool [outdiv5Enable](#)  
*OUTDIV5EN.*

## **KV10Z7**

### **55.48.2.1.0.52 Field Documentation**

**55.48.2.1.0.52.1** `clock_er32k_src_t sim_config_kv10z7_t::er32kSrc`

**55.48.2.1.0.52.2** `uint8_t sim_config_kv10z7_t::outdiv5`

**55.48.2.1.0.52.3** `bool sim_config_kv10z7_t::outdiv5Enable`

### **55.48.3 Macro Definition Documentation**

**55.48.3.1** `#define FTM_EXT_CLK_COUNT 3`

### **55.48.4 Variable Documentation**

**55.48.4.1** `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`



## 55.49 KV30F12810

The data structure definition for KV30F12810 clock manager.

### 55.49.1 Overview

#### Files

- file [fsl\\_clock\\_MKV30F12810.h](#)

#### Data Structures

- 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.*

#### Variables

- uint32\_t [g\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency(FTM\_CLK).*

### 55.49.2 Data Structure Documentation

#### 55.49.2.1 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.*

## **KV30F12810**

### **55.49.2.1.0.53 Field Documentation**

**55.49.2.1.0.53.1** `clock_pllfl_sel_t sim_config_kv30f12810_t::pllFlSel`

**55.49.2.1.0.53.2** `clock_er32k_src_t sim_config_kv30f12810_t::er32kSrc`

**55.49.2.1.0.53.3** `uint8_t sim_config_kv30f12810_t::outdiv4`

### **55.49.3 Macro Definition Documentation**

**55.49.3.1** `#define FTM_EXT_CLK_COUNT 2`

### **55.49.4 Variable Documentation**

**55.49.4.1** `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## 55.50 KV31F12810

The data structure definition for KV31F12810 clock manager.

### 55.50.1 Overview

#### Files

- file [fsl\\_clock\\_MKV31F12810.h](#)

#### Data Structures

- 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.*

#### Variables

- uint32\_t [g\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency(FTM\_CLK).*

### 55.50.2 Data Structure Documentation

#### 55.50.2.1 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.*

## **KV31F12810**

### **55.50.2.1.0.54 Field Documentation**

**55.50.2.1.0.54.1** `clock_pllfl_sel_t sim_config_kv31f12810_t::pllFlSel`

**55.50.2.1.0.54.2** `clock_er32k_src_t sim_config_kv31f12810_t::er32kSrc`

**55.50.2.1.0.54.3** `uint8_t sim_config_kv31f12810_t::outdiv4`

### **55.50.3 Macro Definition Documentation**

**55.50.3.1** `#define FTM_EXT_CLK_COUNT 2`

### **55.50.4 Variable Documentation**

**55.50.4.1** `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## 55.51 KV31F25612

The data structure definition for KV31F25612 clock manager.

### 55.51.1 Overview

#### Files

- file [fsl\\_clock\\_MKV31F25612.h](#)

#### Data Structures

- 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.*

#### Variables

- uint32\_t [g\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency(FTM\_CLK).*

### 55.51.2 Data Structure Documentation

#### 55.51.2.1 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.*

## **KV31F25612**

### **55.51.2.1.0.55 Field Documentation**

**55.51.2.1.0.55.1** `clock_pllfl_sel_t sim_config_kv31f25612_t::pllFlSel`

**55.51.2.1.0.55.2** `clock_er32k_src_t sim_config_kv31f25612_t::er32kSrc`

**55.51.2.1.0.55.3** `uint8_t sim_config_kv31f25612_t::outdiv4`

### **55.51.3 Macro Definition Documentation**

**55.51.3.1** `#define FTM_EXT_CLK_COUNT 2`

### **55.51.4 Variable Documentation**

**55.51.4.1** `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## 55.52 KV31F51212

The data structure definition for KV31F51212 clock manager.

### 55.52.1 Overview

#### Files

- file [fsl\\_clock\\_MKV31F51212.h](#)

#### Data Structures

- 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.*

#### Variables

- uint32\_t [g\\_ftmClkFreq](#) [[FTM\\_EXT\\_CLK\\_COUNT](#)]  
*FTM external clock frequency(FTM\_CLK).*

### 55.52.2 Data Structure Documentation

#### 55.52.2.1 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.*

## **KV31F51212**

### **55.52.2.1.0.56 Field Documentation**

**55.52.2.1.0.56.1** `clock_pllfl_sel_t sim_config_kv31f51212_t::pllFlSel`

**55.52.2.1.0.56.2** `clock_er32k_src_t sim_config_kv31f51212_t::er32kSrc`

**55.52.2.1.0.56.3** `uint8_t sim_config_kv31f51212_t::outdiv4`

### **55.52.3 Macro Definition Documentation**

**55.52.3.1** `#define FTM_EXT_CLK_COUNT 2`

### **55.52.4 Variable Documentation**

**55.52.4.1** `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`



## 55.53 KW21D5

The data structure definition for KW21D5 clock manager.

### 55.53.1 Overview

#### Files

- file [fsl\\_clock\\_MKW21D5.h](#)

#### Data Structures

- struct [sim\\_config\\_kw21d5\\_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).*

### 55.53.2 Data Structure Documentation

#### 55.53.2.1 struct sim\_config\_kw21d5\_t

##### Data Fields

- [clock\\_pllfl\\_sel\\_t](#) [pllFlSel](#)  
*PLL/FLL/IRC48M selection.*

## KW21D5

- `clock_er32k_src_t er32kSrc`  
*ERCLK32K source selection.*
- `uint8_t outdiv4`  
*OUTDIV setting.*

### 55.53.2.1.0.57 Field Documentation

55.53.2.1.0.57.1 `clock_pllfl_sel_t sim_config_kw21d5_t::pllFlSel`

55.53.2.1.0.57.2 `clock_er32k_src_t sim_config_kw21d5_t::er32kSrc`

55.53.2.1.0.57.3 `uint8_t sim_config_kw21d5_t::outdiv4`

### 55.53.3 Macro Definition Documentation

55.53.3.1 `#define USB_EXT_CLK_COUNT 1`

55.53.3.2 `#define FTM_EXT_CLK_COUNT 2`

55.53.3.3 `#define CLOCK_CONFIG_NUM 2`

55.53.3.4 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

55.53.3.5 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

### 55.53.4 Variable Documentation

55.53.4.1 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

55.53.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## 55.54 KW22D5

The data structure definition for KW22D5 clock manager.

### 55.54.1 Overview

#### Files

- file [fsl\\_clock\\_MKW22D5.h](#)

#### Data Structures

- struct [sim\\_config\\_kw22d5\\_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).*

### 55.54.2 Data Structure Documentation

#### 55.54.2.1 struct sim\_config\_kw22d5\_t

##### Data Fields

- [clock\\_pllfl\\_sel\\_t](#) [pllFlSel](#)  
*PLL/FLL/IRC48M selection.*

## KW22D5

- `clock_er32k_src_t er32kSrc`  
*ERCLK32K source selection.*
- `uint8_t outdiv4`  
*OUTDIV setting.*

### 55.54.2.1.0.58 Field Documentation

55.54.2.1.0.58.1 `clock_pllfl_sel_t sim_config_kw22d5_t::pllFlSel`

55.54.2.1.0.58.2 `clock_er32k_src_t sim_config_kw22d5_t::er32kSrc`

55.54.2.1.0.58.3 `uint8_t sim_config_kw22d5_t::outdiv4`

### 55.54.3 Macro Definition Documentation

55.54.3.1 `#define USB_EXT_CLK_COUNT 1`

55.54.3.2 `#define FTM_EXT_CLK_COUNT 2`

55.54.3.3 `#define CLOCK_CONFIG_NUM 2`

55.54.3.4 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

55.54.3.5 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

### 55.54.4 Variable Documentation

55.54.4.1 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

55.54.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## 55.55 KW24D5

The data structure definition for KW24D5 clock manager.

### 55.55.1 Overview

#### Files

- file [fsl\\_clock\\_MKW24D5.h](#)

#### Data Structures

- struct [sim\\_config\\_kw24d5\\_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).*

### 55.55.2 Data Structure Documentation

#### 55.55.2.1 struct sim\_config\_kw24d5\_t

##### Data Fields

- [clock\\_pllfl\\_sel\\_t](#) [pllFlSel](#)  
*PLL/FLL/IRC48M selection.*

## KW24D5

- `clock_er32k_src_t er32kSrc`  
*ERCLK32K source selection.*
- `uint8_t outdiv4`  
*OUTDIV setting.*

### 55.55.2.1.0.59 Field Documentation

55.55.2.1.0.59.1 `clock_pllfl_sel_t sim_config_kw24d5_t::pllFlSel`

55.55.2.1.0.59.2 `clock_er32k_src_t sim_config_kw24d5_t::er32kSrc`

55.55.2.1.0.59.3 `uint8_t sim_config_kw24d5_t::outdiv4`

### 55.55.3 Macro Definition Documentation

55.55.3.1 `#define USB_EXT_CLK_COUNT 1`

55.55.3.2 `#define FTM_EXT_CLK_COUNT 2`

55.55.3.3 `#define CLOCK_CONFIG_NUM 2`

55.55.3.4 `#define CLOCK_CONFIG_INDEX_FOR_VLPR 0`

55.55.3.5 `#define CLOCK_CONFIG_INDEX_FOR_RUN 1`

### 55.55.4 Variable Documentation

55.55.4.1 `uint32_t g_usbClkInFreq[USB_EXT_CLK_COUNT]`

55.55.4.2 `uint32_t g_ftmClkFreq[FTM_EXT_CLK_COUNT]`

## Chapter 56

### Hwtimer\_driver

#### 56.1 Overview

The Kinetis SDK provides the HwTimer driver for various timer modules.

#### Modules

- [HwTimer Driver](#)
- [HwTimer Interrupt handlers](#)

#### Files

- file [fsl\\_hwtimer.h](#)

#### Data Structures

- struct [hwtimer\\_ptr\\_t](#)  
*Hwtimer structure. [More...](#)*
- struct [hwtimer\\_time\\_ptr\\_t](#)  
*Hwtimer\_time structure. [More...](#)*
- struct [hwtimer\\_devif\\_ptr\\_t](#)  
*hwtimer\_devif structure. [More...](#)*

#### Macros

- #define [HWTIMER\\_LL\\_CONTEXT\\_LEN](#) 5U  
*Hwtimer low level context data length definition.*

#### Typedefs

- typedef void(\* [hwtimer\\_callback\\_t](#))(void \*p)  
*Define for low level context data length.*
- typedef [\\_hwtimer\\_error\\_code\\_t](#)(\* [hwtimer\\_devif\\_init\\_t](#))([hwtimer\\_t](#) \*hwtimer, uint32\_t id, void \*data)  
*Type defines init function for devif structure.*
- typedef [\\_hwtimer\\_error\\_code\\_t](#)(\* [hwtimer\\_devif\\_deinit\\_t](#))([hwtimer\\_t](#) \*hwtimer)  
*Type defines deinit function for devif structure.*
- typedef [\\_hwtimer\\_error\\_code\\_t](#)(\* [hwtimer\\_devif\\_set\\_div\\_t](#))([hwtimer\\_t](#) \*hwtimer, uint32\_t period)  
*Type defines set\_div function for devif structure.*
- typedef [\\_hwtimer\\_error\\_code\\_t](#)(\* [hwtimer\\_devif\\_start\\_t](#))([hwtimer\\_t](#) \*hwtimer)  
*Type defines start function for devif structure.*
- typedef [\\_hwtimer\\_error\\_code\\_t](#)(\* [hwtimer\\_devif\\_stop\\_t](#))([hwtimer\\_t](#) \*hwtimer)  
*Type defines stop function for devif structure.*
- typedef [\\_hwtimer\\_error\\_code\\_t](#)(\* [hwtimer\\_devif\\_reset\\_t](#))([hwtimer\\_t](#) \*hwtimer)

## Overview

- Type defines reset function for devif structure.*

• typedef `_hwtimer_error_code_t(* hwtimer_devif_get_time_t)(hwtimer_t *hwtimer, hwtimer_time_t *time)`

*Type defines get\_time function for devif structure.*

## Enumerations

- enum `_hwtimer_error_code_t` {

`kHwtimerSuccess,`

`kHwtimerInvalidInput,`

`kHwtimerInvalidPointer,`

`kHwtimerClockManagerError,`

`kHwtimerRegisterHandlerError,`

`kHwtimerUnknown` }

*Hwtimer error codes definition.*

## Functions

- `_hwtimer_error_code_t HWTIMER_SYS_Init` (`hwtimer_t *hwtimer`, const `hwtimer_devif_t *k-Devif`, `uint32_t id`, `void *data`)

*Initializes caller allocated structure according to given parameters.*
- `_hwtimer_error_code_t HWTIMER_SYS_Deinit` (`hwtimer_t *hwtimer`)

*De-initialization of the hwtimer.*
- `_hwtimer_error_code_t HWTIMER_SYS_SetPeriod` (`hwtimer_t *hwtimer`, `uint32_t period`)

*Set period of hwtimer.*
- `uint32_t HWTIMER_SYS_GetPeriod` (`hwtimer_t *hwtimer`)

*Get period of hwtimer.*
- `_hwtimer_error_code_t HWTIMER_SYS_Start` (`hwtimer_t *hwtimer`)

*Enables the timer and leaves it running.*
- `_hwtimer_error_code_t HWTIMER_SYS_Stop` (`hwtimer_t *hwtimer`)

*The timer stops counting after this function is called.*
- `uint32_t HWTIMER_SYS_GetModulo` (`hwtimer_t *hwtimer`)

*The function returns period of the timer in sub-ticks.*
- `_hwtimer_error_code_t HWTIMER_SYS_GetTime` (`hwtimer_t *hwtimer`, `hwtimer_time_t *time`)

*The function reads the current value of the hwtimer.*
- `uint32_t HWTIMER_SYS_GetTicks` (`hwtimer_t *hwtimer`)

*The function reads the current value of the hwtimer.*
- `_hwtimer_error_code_t HWTIMER_SYS_RegisterCallback` (`hwtimer_t *hwtimer`, `hwtimer_callback_t callbackFunc`, `void *callbackData`)

*Registers function to be called when the timer expires.*
- `_hwtimer_error_code_t HWTIMER_SYS_BlockCallback` (`hwtimer_t *hwtimer`)

*The function is used to block callbacks in circumstances when execution of the callback function is undesired.*
- `_hwtimer_error_code_t HWTIMER_SYS_UnblockCallback` (`hwtimer_t *hwtimer`)

*The function is used to unblock previously blocked callbacks.*
- `_hwtimer_error_code_t HWTIMER_SYS_CancelCallback` (`hwtimer_t *hwtimer`)

*The function cancels pending callback, if any.*



## 56.2 Data Structure Documentation

### 56.2.1 struct hwtimer\_t

This structure defines a hwtimer. The context structure is passed to all API functions (besides other parameters).

#### Warning

Application should not access members of this structure directly.

#### See Also

[HWTIMER\\_SYS\\_init](#)  
[HWTIMER\\_SYS\\_deinit](#)  
[HWTIMER\\_SYS\\_start](#)  
[HWTIMER\\_SYS\\_stop](#)  
[HWTIMER\\_SYS\\_set\\_period](#)  
[HWTIMER\\_SYS\\_get\\_period](#)  
[HWTIMER\\_SYS\\_get\\_modulo](#)  
[HWTIMER\\_SYS\\_get\\_time](#)  
[HWTIMER\\_SYS\\_get\\_ticks](#)  
[HWTIMER\\_SYS\\_callback\\_reg](#)  
[HWTIMER\\_SYS\\_callback\\_block](#)  
[HWTIMER\\_SYS\\_callback\\_unblock](#)  
[HWTIMER\\_SYS\\_callback\\_cancel](#)

### Data Fields

- struct Hwtimer\_devif \* [devif](#)  
*Pointer to device interface structure.*
- uint32\_t [clockFreq](#)  
*Timer's source clock frequency.*
- uint32\_t [divider](#)  
*Actual total divider.*
- uint32\_t [modulo](#)  
*Determine how many sub ticks are in one tick.*
- volatile uint64\_t [ticks](#)  
*Number of elapsed ticks.*
- [hwtimer\\_callback\\_t](#) [callbackFunc](#)  
*Function pointer to be called when the timer expires.*
- void \* [callbackData](#)  
*Arbitrary pointer passed as parameter to the callback function.*
- volatile int [callbackPending](#)  
*Indicate pending callback.*
- int [callbackBlocked](#)  
*Indicate blocked callback.*

## Data Structure Documentation

- `uint32_t llContext [HWTIMER_LL_CONTEXT_LEN]`  
*Private storage locations for arbitrary data keeping the context of the lower layer driver.*

### 56.2.1.0.0.60 Field Documentation

**56.2.1.0.0.60.1** `hwtimer_callback_t hwtimer_ptr_t::callbackFunc`

**56.2.1.0.0.60.2** `void* hwtimer_ptr_t::callbackData`

**56.2.1.0.0.60.3** `volatile int hwtimer_ptr_t::callbackPending`

If the timer overflows when callbacks are blocked the callback becomes pending.

**56.2.1.0.0.60.4** `int hwtimer_ptr_t::callbackBlocked`

**56.2.1.0.0.60.5** `uint32_t hwtimer_ptr_t::llContext[HWTIMER_LL_CONTEXT_LEN]`

### 56.2.2 struct hwtimer\_time\_t

Hwtimer time structure represents a time stamp consisting of timer elapsed periods (TICKS) and current value of the timer counter (subTicks).

See Also

[HWTIMER\\_SYS\\_GetTime](#)

### Data Fields

- `uint64_t ticks`  
*Ticks of timer.*
- `uint32_t subTicks`  
*Sub ticks of timer.*

### 56.2.3 struct hwtimer\_devif\_t

Each low layer driver exports instance of this structure initialized with pointers to API functions the driver implements. The functions themselves should be declared as static (not exported directly).

See Also

[HWTIMER\\_SYS\\_Init](#)  
[HWTIMER\\_SYS\\_Deinit](#)

### Data Fields

- `hwtimer_devif_init_t init`

- *Function pointer to lower layer initialization.*  
[hwtimer\\_devif\\_deinit\\_t deinit](#)
- *Function pointer to lower layer de-initialization.*  
[hwtimer\\_devif\\_set\\_div\\_t setDiv](#)
- *Function pointer to lower layer set divider functionality.*  
[hwtimer\\_devif\\_start\\_t start](#)
- *Function pointer to lower layer start functionality.*  
[hwtimer\\_devif\\_stop\\_t stop](#)
- *Function pointer to lower layer stop functionality.*  
[hwtimer\\_devif\\_get\\_time\\_t getTime](#)
- *Function pointer to lower layer get time functionality.*

## 56.3 Enumeration Type Documentation

### 56.3.1 enum \_hwtimer\_error\_code\_t

Enumerator

- kHwtimerSuccess* success
- kHwtimerInvalidInput* invalid input parameter
- kHwtimerInvalidPointer* invalid pointer
- kHwtimerClockManagerError* clock manager return error
- kHwtimerRegisterHandlerError* Interrupt handler registration returns error.
- kHwtimerUnknown* unknown error

## 56.4 Function Documentation

### 56.4.1 \_hwtimer\_error\_code\_t HWTIMER\_SYS\_Init ( hwtimer\_t \* hwtimer, const hwtimer\_devif\_t \* kDevif, uint32\_t id, void \* data )

The device interface pointer determines low layer driver to be used. Device interface structure is exported by each low layer driver and is opaque to the applications. Please refer to chapter concerning low layer driver below for details. Meaning of the numerical identifier varies depending on low layer driver used. Typically, it identifies particular timer channel to initialize. The initialization function has to be called prior using any other API function.

Parameters

|                |                                                        |
|----------------|--------------------------------------------------------|
| <i>hwtimer</i> | [out] Returns initialized hwtimer structure handle.    |
| <i>kDevif</i>  | [in] Structure determines low layer driver to be used. |
| <i>id</i>      | [in] Numerical identifier of the timer.                |

## Function Documentation

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | [in] Specific data for low level of interrupt. |
|-------------|------------------------------------------------|

### Returns

success or an error code returned from low level init function.

### Return values

|                               |                                                |
|-------------------------------|------------------------------------------------|
| <i>kHwtimerSuccess</i>        | Success                                        |
| <i>kHwtimerInvalidInput</i>   | When input parameter hwtimer is a NULL pointer |
| <i>kHwtimerInvalidPointer</i> | When device structure points to NULL.          |

### Warning

The initialization function has to be called prior using any other API function.

### See Also

HWTIMER\_SYS\_deinit  
HWTIMER\_SYS\_start  
HWTIMER\_SYS\_stop

## 56.4.2 `_hwtimer_error_code_t HWTIMER_SYS_Deinit ( hwtimer_t * hwtimer )`

Calls lower layer stop function to stop timer, then calls low layer de-initialization function and afterwards invalidates hwtimer structure by clearing it.

### Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>hwtimer</i> | [in] Pointer to hwtimer structure. |
|----------------|------------------------------------|

### Returns

success or an error code returned from low level DEINIT function.

### Return values

---

|                               |                                                |
|-------------------------------|------------------------------------------------|
| <i>kHwtimerSuccess</i>        | Success                                        |
| <i>kHwtimerInvalidInput</i>   | When input parameter hwtimer is a NULL pointer |
| <i>kHwtimerInvalidPointer</i> | When device structure points to NULL.          |

See Also

[HWTIMER\\_SYS\\_Init](#)  
[HWTIMER\\_SYS\\_Start](#)  
[HWTIMER\\_SYS\\_Stop](#)

### 56.4.3 `_hwtimer_error_code_t HWTIMER_SYS_SetPeriod ( hwtimer_t * hwtimer, uint32_t period )`

The function provides a way to set up the timer to desired period specified in microseconds. Calls the low layer driver to set up the timer divider according to the specified period.

Parameters

|                |                                                 |
|----------------|-------------------------------------------------|
| <i>hwtimer</i> | [in] Pointer to hwtimer structure.              |
| <i>period</i>  | [in] Required period of timer in micro seconds. |

Returns

success or an error code returned from low level SETDIV function.

Return values

|                                   |                                                                         |
|-----------------------------------|-------------------------------------------------------------------------|
| <i>kHwtimerSuccess</i>            | Success                                                                 |
| <i>kHwtimerInvalidInput</i>       | When input parameter hwtimer or his device structure are NULL pointers. |
| <i>kHwtimerInvalidPointer</i>     | When low level SETDIV function point to NULL.                           |
| <i>kHwtimerClockManager-Error</i> | When Clock manager returns error.                                       |

See Also

[HWTIMER\\_SYS\\_GetPeriod](#)  
[HWTIMER\\_SYS\\_GetModulo](#)  
[HWTIMER\\_SYS\\_GetTime](#)  
[HWTIMER\\_SYS\\_GetTicks](#)

---

## Function Documentation

### 56.4.4 `uint32_t HWTIMER_SYS_GetPeriod ( hwtimer_t * hwtimer )`

The function returns current period of the timer in microseconds calculated from the base frequency and actual divider settings of the timer.

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>hwtimer</i> | [in] Pointer to hwtimer structure. |
|----------------|------------------------------------|

Returns

already set period of hwtimer.

Return values

|   |                                                                         |
|---|-------------------------------------------------------------------------|
| 0 | Input parameter hwtimer is NULL pointer or clock manager returns error. |
|---|-------------------------------------------------------------------------|

See Also

- [HWTIMER\\_SYS\\_SetPeriod](#)
- [HWTIMER\\_SYS\\_GetModulo](#)
- [HWTIMER\\_SYS\\_GetTime](#)
- [HWTIMER\\_SYS\\_GetTicks](#)

### 56.4.5 `_hwtimer_error_code_t HWTIMER_SYS_Start ( hwtimer_t * hwtimer )`

The timer starts counting a new period generating interrupts every time the timer rolls over.

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>hwtimer</i> | [in] Pointer to hwtimer structure. |
|----------------|------------------------------------|

Returns

success or an error code returned from low level START function.

Return values

|                               |                                                |
|-------------------------------|------------------------------------------------|
| <i>kHwtimerSuccess</i>        | Success                                        |
| <i>kHwtimerInvalidInput</i>   | When input parameter hwtimer is a NULL pointer |
| <i>kHwtimerInvalidPointer</i> | When device structure points to NULL.          |

See Also

- [HWTIMER\\_SYS\\_Init](#)
- [HWTIMER\\_SYS\\_Deinit](#)
- [HWTIMER\\_SYS\\_Stop](#)

---

## Function Documentation

### 56.4.6 `_hwtimer_error_code_t HWTIMER_SYS_Stop ( hwtimer_t * hwtimer )`

Pending interrupts and callbacks are cancelled.



Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>hwtimer</i> | [in] Pointer to hwtimer structure. |
|----------------|------------------------------------|

Returns

success or an error code returned from low level STOP function.

Return values

|                               |                                                |
|-------------------------------|------------------------------------------------|
| <i>kHwtimerSuccess</i>        | Success                                        |
| <i>kHwtimerInvalidInput</i>   | When input parameter hwtimer is a NULL pointer |
| <i>kHwtimerInvalidPointer</i> | When device structure points to NULL.          |

See Also

- [HWTIMER\\_SYS\\_Init](#)
- [HWTIMER\\_SYS\\_Deinit](#)
- [HWTIMER\\_SYS\\_Start](#)

### 56.4.7 uint32\_t HWTIMER\_SYS\_GetModulo ( hwtimer\_t \* hwtimer )

It is typically called after [HWTIMER\\_SYS\\_SetPeriod\(\)](#) to obtain actual resolution of the timer in the current configuration.

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>hwtimer</i> | [in] Pointer to hwtimer structure. |
|----------------|------------------------------------|

Returns

resolution of hwtimer.

Return values

|   |                                          |
|---|------------------------------------------|
| 0 | Input parameter hwtimer is NULL pointer. |
|---|------------------------------------------|

See Also

- [HWTIMER\\_SYS\\_SetPeriod](#)
- [HWTIMER\\_SYS\\_GetPeriod](#)
- [HWTIMER\\_SYS\\_GetTime](#)
- [HWTIMER\\_SYS\\_GetTicks](#)

## Function Documentation

### 56.4.8 `_hwtimer_error_code_t HWTIMER_SYS_GetTime ( hwtimer_t * hwtimer, hwtimer_time_t * time )`

Elapsed periods(ticks) and current value of the timer counter (sub-ticks) are filled into the `Hwtimer_time` structure. The sub-ticks number always counts up and is reset to zero when the timer overflows regardless of the counting direction of the underlying device.

Parameters

|                |                                                                                        |
|----------------|----------------------------------------------------------------------------------------|
| <i>hwtimer</i> | [in] Pointer to hwtimer structure.                                                     |
| <i>time</i>    | [out] Pointer to time structure. This value is filled with current value of the timer. |

Returns

success or an error code returned from low level `GET_TIME` function.

Return values

|                               |                                                                                                   |
|-------------------------------|---------------------------------------------------------------------------------------------------|
| <i>kHwtimerSuccess</i>        | Success                                                                                           |
| <i>kHwtimerInvalidInput</i>   | When input parameter <code>hwtimer</code> or input parameter <code>time</code> are NULL pointers. |
| <i>kHwtimerInvalidPointer</i> | When device structure points to NULL.                                                             |

See Also

[HWTIMER\\_SYS\\_SetPeriod](#)  
[HWTIMER\\_SYS\\_GetPeriod](#)  
[HWTIMER\\_SYS\\_GetModulo](#)  
[HWTIMER\\_SYS\\_GetTicks](#)

### 56.4.9 `uint32_t HWTIMER_SYS_GetTicks ( hwtimer_t * hwtimer )`

The returned value corresponds with lower 32 bits of elapsed periods (ticks). The value is guaranteed to be obtained atomically without necessity to mask timer interrupt. Lower layer driver is not involved at all, thus call to this function is considerably faster than `HWTIMER_SYS_GetTime`.

Parameters

---

|                |                                    |
|----------------|------------------------------------|
| <i>hwtimer</i> | [in] Pointer to hwtimer structure. |
|----------------|------------------------------------|

Returns

low 32 bits of 64 bit tick value.

Return values

|   |                                               |
|---|-----------------------------------------------|
| 0 | When input parameter hwtimer is NULL pointer. |
|---|-----------------------------------------------|

See Also

- [HWTIMER\\_SYS\\_SetPeriod](#)
- [HWTIMER\\_SYS\\_GetPeriod](#)
- [HWTIMER\\_SYS\\_GetModulo](#)
- [HWTIMER\\_SYS\\_GetTime](#)

#### 56.4.10 **`_hwtimer_error_code_t HWTIMER_SYS_RegisterCallback ( hwtimer_t * hwtimer, hwtimer_callback_t callbackFunc, void * callbackData )`**

The `callback_data` is arbitrary pointer passed as parameter to the callback function.

Parameters

|                     |                                                                 |
|---------------------|-----------------------------------------------------------------|
| <i>hwtimer</i>      | [in] Pointer to hwtimer structure.                              |
| <i>callbackFunc</i> | [in] Function pointer to be called when the timer expires.      |
| <i>callbackData</i> | [in] Data pointer for the function <code>callback_func</code> . |

Returns

success or an error code

Return values

|                             |                                               |
|-----------------------------|-----------------------------------------------|
| <i>kHwtimerInvalidInput</i> | When input parameter hwtimer is NULL pointer. |
| <i>kHwtimerSuccess</i>      | When registration callback succeed.           |

Warning

This function must not be called from a callback routine.

## Function Documentation

See Also

[HWTIMER\\_SYS\\_BlockCallback](#)  
[HWTIMER\\_SYS\\_UnblockCallback](#)  
[HWTIMER\\_SYS\\_CancelCallback](#)

### 56.4.11 `_hwtimer_error_code_t HWTIMER_SYS_BlockCallback ( hwtimer_t * hwtimer )`

If the timer overflows when callbacks are blocked the callback becomes pending.

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>hwtimer</i> | [in] Pointer to hwtimer structure. |
|----------------|------------------------------------|

Returns

success or an error code

Return values

|                             |                                               |
|-----------------------------|-----------------------------------------------|
| <i>kHwtimerInvalidInput</i> | When input parameter hwtimer is NULL pointer. |
| <i>kHwtimerSuccess</i>      | When callback block succeed.                  |

Warning

This function must not be called from a callback routine.

See Also

[HWTIMER\\_SYS\\_RegCallback](#)  
[HWTIMER\\_SYS\\_UnblockCallback](#)  
[HWTIMER\\_SYS\\_CancelCallback](#)

### 56.4.12 `_hwtimer_error_code_t HWTIMER_SYS_UnblockCallback ( hwtimer_t * hwtimer )`

If there is a callback pending, it gets immediately executed. This function must not be called from a callback routine (it does not make sense to do so anyway as callback function never gets executed while callbacks are blocked).

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>hwtimer</i> | [in] Pointer to hwtimer structure. |
|----------------|------------------------------------|

Returns

success or an error code

Return values

|                             |                                               |
|-----------------------------|-----------------------------------------------|
| <i>kHwtimerInvalidInput</i> | When input parameter hwtimer is NULL pointer. |
| <i>kHwtimerSuccess</i>      | When callback unblock succeed.                |

Warning

This function must not be called from a callback routine.

See Also

- [HWTIMER\\_SYS\\_RegCallback](#)
- [HWTIMER\\_SYS\\_BlockCallback](#)
- [HWTIMER\\_SYS\\_CancelCallback](#)

**56.4.13** `_hwtimer_error_code_t HWTIMER_SYS_CancelCallback ( hwtimer_t * hwtimer )`

Parameters

|                |                                    |
|----------------|------------------------------------|
| <i>hwtimer</i> | [in] Pointer to hwtimer structure. |
|----------------|------------------------------------|

Returns

success or an error code

Return values

|                             |                                               |
|-----------------------------|-----------------------------------------------|
| <i>kHwtimerInvalidInput</i> | When input parameter hwtimer is NULL pointer. |
| <i>kHwtimerSuccess</i>      | When callback cancel succeed.                 |

---

## Function Documentation

### Warning

This function must not be called from a callback routine (it does not make sense to do so anyway as callback function never gets executed while callbacks are blocked).

### See Also

[HWTIMER\\_SYS\\_RegCallback](#)

[HWTIMER\\_SYS\\_BlockCallback](#)

[HWTIMER\\_SYS\\_UnblockCallback](#)

## 56.5 HwTimer Driver

The HwTimer driver provides common APIs for various timer modules.

### 56.5.1 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.

---

## **HwTimer Interrupt handlers**

### **56.6 HwTimer Interrupt handlers**

The HwTimer interrupt handlers provides interrupt handlers for the HW Timers

#### **56.6.1 HwTimer Interrupt handler overview**

The HwTimer interrupt handlers are not included in the SDK library. The user has to include them when building the application.



## Chapter 57

# Interrupt Manager (Interrupt)

The Kinetis SDK Interrupt Manager provides a set of API/services to configure the Interrupt Controller (NVIC).

### 57.1 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.*

#### 57.1.1 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\_IRQn 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);
```

## 57.2 Enumeration Type Documentation

### 57.2.1 enum interrupt\_status\_t

## 57.3 Function Documentation

### 57.3.1 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

### 57.3.2 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 |
|------------------|------------|

**57.3.3 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 |
|------------------|------------|

**57.3.4 void INT\_SYS\_EnableIRQGlobal ( void )**

This function enables the global interrupt by calling the core API.

**57.3.5 void INT\_SYS\_DisableIRQGlobal ( void )**

This function disables the global interrupt by calling the core API.



## Chapter 58 Power Manager (Power)

### 58.1 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,  
kPowerManagerVlprw,  
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 `**power-ConfigsPtr`, `uint8_t` `configsNumber`, `power_manager_callback_user_config_t` `**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` const `**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_GetErrorCallbackIndex` (`void`)  
*This function returns the last failed notification callback.*
- `power_manager_callback_user_config_t * POWER_SYS_GetErrorCallback` (`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.*

## 58.2 Data Structure Documentation

### 58.2.1 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.

## Data Structure Documentation

### Data Fields

- [power\\_manager\\_modes\\_t mode](#)  
*Power mode.*
- bool [sleepOnExitValue](#)  
*Sleep or deep sleep after interrupt service finished.*

### 58.2.2 struct power\_manager\_notify\_struct\_t

#### Data Fields

- uint8\_t [targetPowerConfigIndex](#)  
*Target power configuration index.*
- [power\\_manager\\_user\\_config\\_t](#) const \* [targetPowerConfigPtr](#)  
*Pointer to target power configuration.*
- [power\\_manager\\_policy\\_t](#) [policy](#)  
*Clock transition policy.*
- [power\\_manager\\_notify\\_t](#) [notifyType](#)  
*Clock notification type.*

#### 58.2.2.0.0.61 Field Documentation

58.2.2.0.0.61.1 uint8\_t power\_manager\_notify\_struct\_t::targetPowerConfigIndex

58.2.2.0.0.61.2 power\_manager\_policy\_t power\_manager\_notify\_struct\_t::policy

58.2.2.0.0.61.3 power\_manager\_notify\_t power\_manager\_notify\_struct\_t::notifyType

### 58.2.3 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.

### 58.2.4 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](#)  
const \*\* [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.*

### 58.2.4.0.0.62 Field Documentation

**58.2.4.0.0.62.1** [power\\_manager\\_user\\_config\\_t](#) const\*\* [power\\_manager\\_state\\_t::configs](#)

**58.2.4.0.0.62.2** [power\\_manager\\_callback\\_user\\_config\\_t](#)\*\* [power\\_manager\\_state\\_t::staticCallbacks](#)

**58.2.4.0.0.62.3** [uint8\\_t power\\_manager\\_state\\_t::staticCallbacksNumber](#)

number of callback configurations

**58.2.4.0.0.62.4** [uint8\\_t power\\_manager\\_state\\_t::errorCallbackIndex](#)

**58.2.4.0.0.62.5** [uint8\\_t power\\_manager\\_state\\_t::currentConfig](#)

## 58.3 Typedef Documentation

### 58.3.1 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.

**58.3.2** `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\\_](#)

## Enumeration Type Documentation

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\(\)](#)).

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](#).

## 58.4 Enumeration Type Documentation

### 58.4.1 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:

- [kPowerManagerHsr](#) - 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.

### 58.4.2 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.

### 58.4.3 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.

### 58.4.4 enum power\_manager\_notify\_t

Used to notify registered callbacks

Enumerator

- kPowerManagerNotifyRecover* Notify IP to recover to previous work state.

## Function Documentation

*kPowerManagerNotifyBefore* Notify IP that system changes power setting.

*kPowerManagerNotifyAfter* Notify IP that have changed to new power setting.

### 58.4.5 enum power\_manager\_callback\_type\_t

Used in the callback configuration structures ([power\\_manager\\_callback\\_user\\_config\\_t](#)) to specify when the registered callback is 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.

## 58.5 Function Documentation

### 58.5.1 power\_manager\_error\_code\_t POWER\_SYS\_Init ( power\_manager\_user\_config\_t const \*\* powerConfigsPtr, uint8\_t configsNumber, power\_manager\_callback\_user\_config\_t \*\* 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 are 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 = {
 kPowerManagerV1pw,
 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>powerConfigsPtr</i> | A pointer to an array with references to all power configurations which is handled by the 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>callbacksNumber</i> | Number of registered callbacks. Size of callbacksPtr array.                                                                                                  |

Returns

An error code or kPowerManagerSuccess.

**58.5.2 power\_manager\_error\_code\_t POWER\_SYS\_Deinit ( void )**

## Function Documentation

Returns

An error code or `kPowerManagerSuccess`.

### 58.5.3 `power_manager_error_code_t POWER_SYS_SetMode ( uint8_t powerModeIndex, 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>powerMode-Index</i> | Requested power mode represented as an index into array of user-defined power mode configurations passed to the <code>POWER_SYS_Init()</code> . |
| <i>policy</i>          | Transaction policy                                                                                                                              |

Returns

An error code or `kPowerManagerSuccess`.

#### 58.5.4 `power_manager_error_code_t POWER_SYS_GetLastMode ( uint8_t * powerModeIndexPtr )`

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>powerModeIndexPtr</i> | Power mode which has been set represented as an index into array of power mode configurations passed to the <a href="#">POWER_SYS_Init()</a> . |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|

Returns

An error code or `kPowerManagerSuccess`.

#### 58.5.5 `power_manager_error_code_t POWER_SYS_GetLastModeConfig ( power_manager_user_config_t const ** 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`.

#### 58.5.6 `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 `kPowerManagerHsrn` value.

Returns

Currently used run power mode.

## Function Documentation

### 58.5.7 `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.

Returns

Callback index of last failed callback or value equal to callbacks count.

### 58.5.8 `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.

### 58.5.9 `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.

### 58.5.10 `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.



### 58.5.11 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.

### 58.5.12 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

### 58.6 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.

#### 58.6.1 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}

#include "fsl_power_manager.h"

/* Definition of power manager callback */
power_manager_error_code_t callback0(power_manager_callback_type_t type,
    power_manager_user_config_t * configPtr,
    power_manager_callback_data_t * dataPtr)
{
    power_manager_error_code_t ret = kPowerManagerError;

    ...
    ...
    ...

    return ret;
}

...
...
...
...
...
/* Main function */
int main(void)
{
    /* Callback configuration */
    user_callback_data_t callbackData0;

    power_manager_static_callback_user_config_t callbackCfg0 = { callback0,
        kPowerManagerCallbackBeforeAfter,
        (power_manager_callback_data_t*) &callbackData0 };

    power_manager_static_callback_user_config_t * callbacks[] =
        { &callbackCfg0 };
}
```

```

/* 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}

```



---

## Power Manager driver

## Chapter 59

### Utilities for the Kinetis SDK

This part describes the programming interface of the debug console driver.

#### 59.1 Debug Console Initialization

To initialize the DbgConsole module, call the DbgConsole\_Init() function and pass in the user configuration structure. This function automatically enables the module and clock. After the DbgConsole\_Init() function is called and returned, stdout and stdin will be connected to the selected UART/LPUART.

Pass a user configuration debug\_console\_device\_type\_t shown here:

```
typedef enum _debug_console_device_type {
 kDebugConsoleNone = 0U,
 kDebugConsoleLPSCI = 15U, /*<! Use strange start number to avoid treating 0
 as correct device type. Sometimes user forget
 to specify the device type but only use the
 default value '0' as the device type. */

 kDebugConsoleUART = 16U,
 kDebugConsoleLPUART = 17U
} debug_console_device_type_t;
```

Debug console state is stored in debug\_console\_state\_t structure:

```
typedef struct DebugConsoleState {
 debug_console_device_type_t type; /*<! Indicator telling whether the debug console is initied. */
 uint8_t instance; /*<! Instance number indicator. */
 uint32_t baseAddr;
 debug_console_ops_t ops; /*<! Operation function pointers for debug uart operations. */
} debug_console_state_t;
```

This example shows how to call the DbgConsole\_Init() given the user configuration structure and the instance number.

```
DbgConsole_Init(BOARD_DEBUG_UART_INSTANCE, BOARD_DEBUG_UART_BAUD, kDebugConsoleUART);
```

#### Debug Console formatted IO

Debug console has its own printf/scanf/putchar/getchar functions which are defined in the header:

```
int debug_printf(const char *fmt_s, ...);
int debug_putchar(int ch);
int debug_scanf(const char *fmt_ptr, ...);
int debug_getchar(void);
```

Choose toolchain's printf/scanf or KSDK version printf/scanf:

## Debug Console Initialization

```
#if (defined (FSL_RTOS_MQX) && (MQX_COMMON_CONFIG != MQX_LITE_CONFIG))
#define PRINTF printf
#define SCANF scanf
#define PUTCHAR putchar
#define GETCHAR getchar
#else
/*Configuration for toolchain's printf/scanf or KSDK version printf/scanf */
#define PRINTF debug_printf
//#define PRINTF printf
#define SCANF debug_scanf
//#define SCANF scanf
#define PUTCHAR debug_putchar
//#define PUTCHAR putchar
#define GETCHAR debug_getchar
//#define GETCHAR getchar
#endif
```

Print out failure messages using KSDK's `assert_func`:

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

 for (;;)
 {}
}
```

Function `_doprint` outputs its parameters according to a formatted string. I/O is performed by calling given function pointer using `(*func_ptr)(c,farg)`.

```
int _doprint(void *farg, PUTCHAR_FUNC func_ptr, int max_count, char *fmt, va_list ap)
```

Function `scan_prv` converts an input line of ASCII characters based upon a provided string format.

```
int scan_prv(const char *line_ptr, char *format, va_list args_ptr)
```

Function `mknumstr` converts a radix number to a string and return its length.

```
static int32_t mknumstr (char *numstr, void *nump, int32_t neg, int32_t radix, bool use_caps);
```

Function `mkfloatnumstr` converts a floating radix number to a string and return its length.

```
static int32_t mkfloatnumstr (char *numstr, void *nump, int32_t radix, uint32_t precision_width);
```

## Chapter 60

# OS Abstraction Layer (OSA)

The Kinetis SDK provides the timer services for all OSA layers.

### 60.1 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 RTOS Abstraction Layer](#)  
*The Kinetis SDK provides the MQX RTOS 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.*

#### Typedefs

- typedef void(\* [osa\\_int\\_handler\\_t](#))(void)  
*OSA interrupt handler.*

#### 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.*

## Overview

### Counting Semaphore

- [osa\\_status\\_t OSA\\_SemaCreate](#) ([semaphore\\_t](#) \*pSem, [uint8\\_t](#) initValue)  
*Creates a semaphore with a given value.*
- [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_int_handler_t OSA_InstallIntHandler (int32_t IRQNumber, osa_int_handler_t handler)`  
*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.*

## Overview

### 60.1.1 OS Abstraction Layer

#### 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,  $\mu$ C/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,  $\mu$ C/OS-II and  $\mu$ C/OS-III need pre-allocated task stack while other RTOSes do not need this. The  $\mu$ C/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 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 RTOS, 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.

## Overview

Note that if multiple tasks are waiting for one semaphore, different RTOSes may have different behaviors, for example, wake up the task wait first or wake up the task with highest priority. Bare metal semaphore does not support multiple tasks wait with timeout.

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

When using event, please notice that if multiple tasks are waiting on one event, different RTOSes may have different behaviors. For example, only the first task in waiting list is woke up, or all tasks in waiting list are woke up. Bare metal event does not support multiple tasks wait with timeout.

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 if multiple tasks are waiting on one message queue, different RTOSes may have different behaviors. Bare metal message queue does not support multiple tasks wait with timeout.

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.

## Overview

If the queue is not used any more, use the [OSA\\_MsgQDestroy\(\)](#) function to destroy it.

## 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 RTOS, 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\(\)](#). Please call [OSA\\_Init\(\)](#) after [hardware\\_init\(\)](#).

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
{
 hardware_init();

 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.

## 60.2 Typedef Documentation

### 60.2.1 typedef void(\* osa\_int\_handler\_t)(void)

## 60.3 Enumeration Type Documentation

### 60.3.1 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.

### 60.3.2 enum osa\_event\_clear\_mode\_t

Enumerator

*kEventAutoClear* The flags of the event will be cleared automatically.

## Function Documentation

*kEventManualClear* The flags of the event will be cleared manually.

### 60.3.3 enum osa\_critical\_section\_mode\_t

Enumerator

*kCriticalLockSched* Lock scheduler in critical part.

*kCriticalDisableInt* Disable interrupt in critical selection.

## 60.4 Function Documentation

### 60.4.1 osa\_status\_t OSA\_SemaCreate ( semaphore\_t \* pSem, uint8\_t initValue )

This function creates a semaphore and sets the value to the parameter initValue.

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 cannot be created.       |

Example:

```
semaphore_t mySem;
OSA_SemaCreate (&mySem, 0);
```

### 60.4.2 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\_OSA\_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 OSA_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 cannot be waited on by more than one task at the same time.

**Example:**

```
osa_status_t status;
status = OSA_SemaWait (&mySem, 100);
switch(status)
{
 //...
}
```

**60.4.3 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 cannot be signaled or invalid parameter. |

**Example:**

```
osa_status_t status;
status = OSA_SemaPost (&mySem);
switch(status)
{
 //...
}
```

---

## Function Documentation

**60.4.4** `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 cannot be destroyed.       |

Example:

```

osa_status_t status;
status = OSA_SemaDestroy (&mySem);
switch(status)
{
 //...
}

```

### 60.4.5 `osa_status_t OSA_MutexCreate ( mutex_t * pMutex )`

This function creates a non-recursive mutex and sets it to unlocked status.

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 cannot be created.       |

Example:

```

mutex_t myMutex;
osa_status_t status;
status = OSA_MutexCreate (&myMutex);
switch (status)
{
 //...
}

```

### 60.4.6 `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.

## Function Documentation

### 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 cannot try to lock the mutex it has locked.

### Example:

```
osa_status_t status;
status = OSA_MutexLock(&myMutex, 100);
switch (status)
{
 //...
}
```

## 60.4.7 `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 cannot be unlocked or invalid parameter. |
|--------------------------|----------------------------------------------------|

Example:

```
osa_status_t status;
status = OSA_MutexUnlock (&myMutex);
switch (status)
{
 //...
}
```

### 60.4.8 **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 cannot be destroyed.       |

Example:

```
osa_status_t status;
status = OSA_MutexDestroy (&myMutex);
switch (status)
{
 //...
}
```

### 60.4.9 **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.   |

## Function Documentation

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);
```

### 60.4.10 **osa\_status\_t OSA\_EventWait ( event\_t \* pEvent, event\_flags\_t flagsToWait, bool waitAll, 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>waitAll</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 cannot be waited on by more than one tasks at the same time.
  1. Please pay attention to the flags bit width, FreeRTOS uses the most significant 8 bis 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)
{
 //...
}

```

### 60.4.11 `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)
{
 //...
}

```

### 60.4.12 `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.    |

## Function Documentation

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)
{
 //...
}
```

### 60.4.13 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);
```

### 60.4.14 osa\_status\_t OSA\_EventDestroy ( event\_t \* pEvent )

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)
{
 //...
}

```

**60.4.15 `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 cannot 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)
{
 //...
}

```

## Function Documentation

### 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.

### 60.4.16 `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)
{
 //...
}
```

### 60.4.17 `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.

#### Return values

|                            |                                      |
|----------------------------|--------------------------------------|
| <i>kStatus_OSA_Success</i> | The function is called successfully. |
| <i>kStatus_OSA_Error</i>   | Error occurs with this function.     |

#### Example:

```
osa_status_t status;
status = OSA_TaskYield();
switch (status)
{
 //...
}
```

### 60.4.18 `task_handler_t OSA_TaskGetHandler ( void )`

Returns

Handler to current active task.

Example:

```
task_handler_t handler = OSA_TaskYield();
```

### 60.4.19 uint16\_t OSA\_TaskGetPriority ( task\_handler\_t handler )

Parameters

|                |                                                     |
|----------------|-----------------------------------------------------|
| <i>handler</i> | The handler of the task whose priority is received. |
|----------------|-----------------------------------------------------|

Returns

Task's priority.

Example:

```
uint16_t taskPrio = OSA_TaskGetPriority(taskHandler);
```

### 60.4.20 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 cannot be set.       |

Example:

```
osa_status_t status;
status = OSA_TaskSetPriority(taskHandler, newPrio);
switch (status)
{
 //...
}
```

## Function Documentation

### 60.4.21 `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 the use of the function:

```
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.

### 60.4.22 `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`;

#### 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

|                                  |                                                        |
|----------------------------------|--------------------------------------------------------|
| <code>kStatus_OSA_Success</code> | Message successfully put into the queue.               |
| <code>kStatus_OSA_Error</code>   | 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)
{
 //...
}
```

**60.4.23** `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.

## Function Documentation

### 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)
{
 //...
}
```

## 60.4.24 `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)
{
 //...
}
```

### 60.4.25 void\* OSA\_MemAlloc ( 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.

### 60.4.26 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.

### 60.4.27 osa\_status\_t OSA\_MemFree ( void \* ptr )

Parameters

|            |                                                               |
|------------|---------------------------------------------------------------|
| <i>ptr</i> | Pointer to the start of the memory block previously reserved. |
|------------|---------------------------------------------------------------|

## Function Documentation

Return values

|                            |                                      |
|----------------------------|--------------------------------------|
| <i>kStatus_OSA_Success</i> | Memory correctly freed.              |
| <i>kStatus_OSA_Error</i>   | Error occurs during free the memory. |

### 60.4.28 void OSA\_TimeDelay ( uint32\_t *delay* )

Parameters

|              |                                   |
|--------------|-----------------------------------|
| <i>delay</i> | The time in milliseconds to wait. |
|--------------|-----------------------------------|

### 60.4.29 uint32\_t OSA\_TimeGetMsec ( void )

Returns

Current time in milliseconds.

### 60.4.30 osa\_int\_handler\_t OSA\_InstallIntHandler ( int32\_t *IRQNumber*, osa\_int\_handler\_t *handler* )

Parameters

|                  |                                   |
|------------------|-----------------------------------|
| <i>IRQNumber</i> | IRQ number of the interrupt.      |
| <i>handler</i>   | The interrupt handler to install. |

Returns

This function returns the old interrupt handler installed in vector table. If could not install ISR, this function returns NULL; The return value could be compared with OSA\_DEFAULT\_INT\_HANDLER to detect whether this is the first interrupt handler installed.

### 60.4.31 void OSA\_EnterCritical ( osa\_critical\_part\_mode\_t *mode* )



Parameters

|             |                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>mode</i> | Lock task scheduler of disable interrupt in critical part. Pass kCriticalLockSched to lock task scheduler, pass kCriticalDisableInt to disable interrupt. |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|

**60.4.32 void OSA\_ExitCritical ( osa\_critical\_part\_mode\_t mode )**

Parameters

|             |                                                                                                                                                           |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>mode</i> | Lock task scheduler of disable interrupt in critical part. Pass kCriticalLockSched to lock task scheduler, pass kCriticalDisableInt to disable interrupt. |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|

**60.4.33 osa\_status\_t OSA\_Init ( void )**

This function sets up the basic RTOS services. It should be called first in the main function.

Return values

|                            |                                             |
|----------------------------|---------------------------------------------|
| <i>kStatus_OSA_Success</i> | RTOS services are initialized successfully. |
| <i>kStatus_OSA_Error</i>   | Error occurs during initialization.         |

**60.4.34 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.    |

## Bare Metal Abstraction Layer

### 60.5 Bare Metal Abstraction Layer

The Kinetis SDK provides the Bare Metal Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

#### 60.5.1 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 [FSL\\_OSA\\_BM\\_TIMER\\_NONE](#) 0U  
*Bare Metal does not use timer.*
- #define [FSL\\_OSA\\_BM\\_TIMER\\_LPTMR](#) 1U  
*Bare Metal uses LPTMR as timer.*
- #define [FSL\\_OSA\\_BM\\_TIMER\\_CONFIG](#) [FSL\\_OSA\\_BM\\_TIMER\\_LPTMR](#)  
*Configure what timer is used in Bare Metal.*
- #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.*
- #define [OSA\\_DEFAULT\\_INT\\_HANDLER](#) (([osa\\_int\\_handler\\_t](#))(&[DefaultISR](#)))  
*The default interrupt handler installed in vector table.*

#### 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.*

## Functions

- void `DefaultISR` (void)  
*The default interrupt handler installed in vector table.*

## 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.*

### 60.5.1.1 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 Abstraction Layer

### 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)
 {
 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 Mutex

Bare metal OSA implements mutex as a binary semaphore, this is different from RTOSes mutex. Applications could choose to use it or not for bare metal.

## Bare Metal Time management

Bare metal OSA implements two configurations for time management. The first one is using lowpower timer. The second one is empty, in other words, this configuration disables time management in bare metal OSA. To use different configurations, please set the macro `FSL_OSA_BM_TIMER_CONFIG` in file `fsl_os_abstraction_bm.h`.

### Time management with LPTMR

To use lowpower timer in bare metal OSA, please define `FSL_OSA_BM_TIMER_CONFIG` as `FSL_OSA_BM_TIMER_LPTMER`.

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

## Bare Metal Abstraction Layer

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()` cannot delay longer than 65536ms.
2. Wait functions, such as `OSA_SemaWait()`, cannot set timeout longer than 65536ms, however, `OSA_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.

### Disable time management in BM OSA

To disable time management bare metal OSA, please define `FSL_OSA_BM_TIMER_CONFIG` as `FSL_OSA_BM_TIMER_NONE`.

With this configuration, LPTMR is not used by BM OSA, then the footprint is smaller. Time services such `OSA_TimeGetMsec`, `OSA_TimeDelay` could not be used any more. At the same time, the wait functions such as `OSA_SemaWait()`, `OSA_EventWait()` could only use 0 or `OSA_WAIT_FOREVER` as the parameter timeout.

### User defined time management

Sometimes the LPTMR should be used for the other purpose, and also BM OSA should provide time services. In this situation, BM OSA must use other timers for the time services, please re-write these functions defined in `fsl_os_abstraction_bm.c`:

```
/* Initialize timer. */
void OSA_TimeInit(void);

/*
 * Get time delta between time_start and time_end. This function
 * should consider the timer counter overflow.
 */
uint32_t OSA_TimeDiff(uint32_t time_start, uint32_t time_end);

/* Get current time in milliseconds. */
uint32_t OSA_TimeGetMsec(void);
```

There are two methods to re-write these functions:

1. Modify the functions in `fsl_os_abstraction_bm.c` directly.
2. Define the functions in application, then the functions in `fsl_os_abstraction_bm.c` will be overridden.

## 60.5.2 Data Structure Documentation

### 60.5.2.1 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.*

### 60.5.2.2 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.*

#### 60.5.2.2.0.63 Field Documentation

##### 60.5.2.2.0.63.1 LWSEM\_STRUCT mutex\_t::sema

##### 60.5.2.2.0.63.2 \_task\_id mutex\_t::owner

### 60.5.2.3 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.*

## Bare Metal Abstraction Layer

### 60.5.2.4 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.*

### 60.5.2.5 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.*



### 60.5.3 Macro Definition Documentation

60.5.3.1 #define FSL\_OSA\_BM\_TIMER\_NONE 0U

60.5.3.2 #define FSL\_OSA\_BM\_TIMER\_LPTIMER 1U

60.5.3.3 #define FSL\_OSA\_BM\_TIMER\_CONFIG FSL\_OSA\_BM\_TIMER\_LPTIMER

60.5.3.4 #define OSA\_WAIT\_FOREVER 0xFFFFFFFFU

60.5.3.5 #define TASK\_MAX\_NUM 5

60.5.3.6 #define FSL\_OSA\_TIME\_RANGE 0xFFFFU

60.5.3.7 #define OSA\_DEFAULT\_INT\_HANDLER ((osa\_int\_handler\_t)&DefaultISR)

60.5.3.8 #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

|                  |                                          |
|------------------|------------------------------------------|
| <i>task</i>      | The task function.                       |
| <i>stackSize</i> | The stack size this task needs in bytes. |

60.5.3.9 #define MSG\_QUEUE\_DECLARE( *name*, *number*, *size* )

**Value:**

```
uint32_t queueMem_##name[number * size]; \
entity_##name = { \
 .queueMem = queueMem_##name \
}; \
*name = &(entity_##name) \
msg_queue_t
```

## Bare Metal Abstraction Layer

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.   |

### 60.5.4 Function Documentation

#### 60.5.4.1 void DefaultISR ( void )

#### 60.5.4.2 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.

## 60.6 MQX RTOS Abstraction Layer

The Kinetis SDK provides the MQX RTOS Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

### 60.6.1 Overview

#### Data Structures

- struct [mutex\\_t](#)  
*Type for a mutex. [More...](#)*

#### Macros

- #define [OSA\\_WAIT\\_FOREVER](#) 0xFFFFFFFFU  
*Constant to pass as timeout value in order to wait indefinitely.*
- #define [FSL\\_OSA\\_TIME\\_RANGE](#) 0xFFFFFFFFU  
*OSA's time range in millisecond, OSA time wraps if exceeds this value.*
- #define [OSA\\_DEFAULT\\_INT\\_HANDLER](#) ([OSA\\_DefaultIntHandler](#)())  
*The default interrupt handler installed in vector table.*
- #define [MQX\\_MAIN\\_TASK\\_PRIORITY](#) (7+16)  
*The priority of MQX RTOS Main\_Task.*

#### Typedefs

- typedef LWSEM\_STRUCT [semaphore\\_t](#)  
*Type for MQX RTOS 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.*

## MQX RTOS Abstraction Layer

### Functions

- static `osa_int_handler_t OSA_DefaultIntHandler` (void)  
*The default interrupt handler installed in vector table.*

### Thread management

- #define `OSA_TASK_DEFINE`(task, stackSize)  
*Defines a task.*
- #define `PRIORITY_OSA_TO_RTOS`(osa\_prio) ((osa\_prio)+7U)  
*To provide unified task priority for upper layer, OSA layer makes conversion.*
- #define `PRIORITY_RTOS_TO_OSA`(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.*

## 60.6.2 Data Structure Documentation

### 60.6.2.1 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.*

60.6.2.1.0.64 Field Documentation

60.6.2.1.0.64.1 LWSEM\_STRUCT mutex\_t::sema

60.6.2.1.0.64.2 \_task\_id mutex\_t::owner

60.6.3 Macro Definition Documentation

60.6.3.1 #define OSA\_WAIT\_FOREVER 0xFFFFFFFFU

60.6.3.2 #define FSL\_OSA\_TIME\_RANGE 0xFFFFFFFFU

60.6.3.3 #define OSA\_DEFAULT\_INT\_HANDLER (OSA\_DefaultIntHandler())

60.6.3.4 #define MQX\_MAIN\_TASK\_PRIORITY (7+16)

60.6.3.5 #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. |

60.6.3.6 #define PRIORITY\_OSA\_TO\_RTOS( *osa\_prio* ) ((*osa\_prio*)+7U)

MQX RTOS highest 7 priorities are special priorities.

60.6.3.7 #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*]

## MQX RTOS Abstraction Layer

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.      |

### 60.6.4 Typedef Documentation

#### 60.6.4.1 typedef LWSEM\_STRUCT semaphore\_t

Type for a semaphore.

#### 60.6.4.2 typedef LWEVENT\_STRUCT event\_t

#### 60.6.4.3 typedef \_mqx\_uint event\_flags\_t

#### 60.6.4.4 typedef TASK\_FPTR task\_t

#### 60.6.4.5 typedef \_task\_id task\_handler\_t

#### 60.6.4.6 typedef uint32\_t task\_param\_t

#### 60.6.4.7 typedef uint32\_t task\_stack\_t

#### 60.6.4.8 typedef \_mqx\_max\_type msg\_queue\_t

#### 60.6.4.9 typedef void\* msg\_queue\_handler\_t

### 60.6.5 Function Documentation

#### 60.6.5.1 static osa\_int\_handler\_t OSA\_DefaultIntHandler ( void ) [inline], [static]

## 60.7 μC/OS-II Abstraction Layer

The Kinetis SDK provides the μC/OS-II Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

### 60.7.1 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` `0xFFFFFFFFU`  
*Constant to pass as timeout value to wait indefinitely.*
- #define `FSL_OSA_TIME_RANGE` `0xFFFFFFFFU`  
*OSA's time range in millisecond, OSA time wraps if exceeds this value.*
- #define `OSA_DEFAULT_INT_HANDLER` `((osa_int_handler_t)&DefaultISR)`  
*The default interrupt handler installed in vector table.*

#### 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.*

## μC/OS-II Abstraction Layer

### Thread management

- #define `OSA_TASK_DEFINE`(task, stackSize)  
*Defines a task.*
- #define `PRIORITY_OSA_TO_RTOS`(osa\_prio) ((osa\_prio)+4U)  
*To provide unified task priority for upper layer, OSA layer makes conversion.*
- #define `PRIORITY_RTOS_TO_OSA`(rtos\_prio) ((rtos\_prio)-4U)

### Message queues

- #define `MSG_QUEUE_DECLARE`(name, number, size)  
*This macro statically reserves the memory required for the queue.*

## 60.7.2 Data Structure Documentation

### 60.7.2.1 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.*

### 60.7.2.2 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.*



### 60.7.3 Macro Definition Documentation

60.7.3.1 #define OSA\_WAIT\_FOREVER 0xFFFFFFFFU

60.7.3.2 #define FSL\_OSA\_TIME\_RANGE 0xFFFFFFFFU

60.7.3.3 #define OSA\_DEFAULT\_INT\_HANDLER ((osa\_int\_handler\_t)(&DefaultISR))

60.7.3.4 #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. |

60.7.3.5 #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.  |

## μC/OS-II Abstraction Layer

|             |                                  |
|-------------|----------------------------------|
| <i>size</i> | Size of every elements in words. |
|-------------|----------------------------------|

## 60.8 μC/OS-III Abstraction Layer

The Kinetis SDK provides the μC/OS-III Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

### 60.8.1 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](#) 0xFFFFFFFFU  
*Constant to pass as timeout value in order to wait indefinitely.*
- #define [FSL\\_OSA\\_TIME\\_RANGE](#) 0xFFFFFFFFU  
*OSA's time range in millisecond, OSA time wraps if exceeds this value.*
- #define [OSA\\_DEFAULT\\_INT\\_HANDLER](#) ((osa\_int\_handler\_t)(&DefaultISR))  
*The default interrupt handler installed in vector table.*

#### 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.*

## μC/OS-III Abstraction Layer

### 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.*

## 60.8.2 Data Structure Documentation

### 60.8.2.1 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.*

### 60.8.2.2 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*

### 60.8.3 Macro Definition Documentation

**60.8.3.1 #define OSA\_WAIT\_FOREVER 0xFFFFFFFFU**

**60.8.3.2 #define FSL\_OSA\_TIME\_RANGE 0xFFFFFFFFU**

**60.8.3.3 #define OSA\_DEFAULT\_INT\_HANDLER ((osa\_int\_handler\_t)(&DefaultISR))**

**60.8.3.4 #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. |

**60.8.3.5 #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.

**60.8.3.6 #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)
```

## μC/OS-III Abstraction Layer

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.  |

### 60.8.4 Typedef Documentation

60.8.4.1 typedef OS\_TCB\* task\_handler\_t

60.8.4.2 typedef CPU\_STK task\_stack\_t

60.8.4.3 typedef OS\_SEM semaphore\_t

60.8.4.4 typedef OS\_MUTEX mutex\_t

60.8.4.5 typedef OS\_FLAGS event\_flags\_t

60.8.4.6 typedef event\_ucosiii event\_t

60.8.4.7 typedef msgq\_struct\_ucosiii msg\_queue\_t

60.8.4.8 typedef msg\_queue\_t\* msg\_queue\_handler\_t

## 60.9 FreeRTOS Abstraction Layer

The Kinetis SDK provides the FreeRTOS Abstraction Layer for synchronization, mutual exclusion, message queue, etc.

### 60.9.1 Overview

#### Data Structures

- struct [event\\_freertos](#)  
*Type for an event group object in FreeRTOS. [More...](#)*

#### Macros

- #define [OSA\\_WAIT\\_FOREVER](#) 0xFFFFFFFFU  
*Constant to pass as timeout value in order to wait indefinitely.*
- #define [FSL\\_OSA\\_TIME\\_RANGE](#) 0xFFFFFFFFU  
*OSA's time range in millisecond, OSA time wraps if exceeds this value.*
- #define [OSA\\_DEFAULT\\_INT\\_HANDLER](#) ((osa\_int\_handler\_t)(&DefaultISR))  
*The default interrupt handler installed in vector table.*

#### 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.*

## FreeRTOS Abstraction Layer

### Thread management

- #define `OSA_TASK_DEFINE(task, stackSize)`  
*Creates a task descriptor that is used to create the task with `OSA_TaskCreate`.*
- #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.*

## 60.9.2 Data Structure Documentation

### 60.9.2.1 struct event\_freertos

#### Data Fields

- EventGroupHandle\_t `eventHandler`  
*FreeRTOS event handler.*
- `osa_event_clear_mode_t` `clearMode`  
*Auto clear or manual clear.*

## 60.9.3 Macro Definition Documentation

60.9.3.1 #define `OSA_WAIT_FOREVER` `0xFFFFFFFFFU`

60.9.3.2 #define `FSL_OSA_TIME_RANGE` `0xFFFFFFFFFU`

60.9.3.3 #define `OSA_DEFAULT_INT_HANDLER` `((osa_int_handler_t)(&DefaultISR))`

60.9.3.4 #define `OSA_TASK_DEFINE( task, stackSize )`

#### Value:

```
task_stack_t* task##_stack = NULL; \
task_handler_t task##_task_handler
```



Parameters

|                  |                                                |
|------------------|------------------------------------------------|
| <i>task</i>      | The task function.                             |
| <i>stackSize</i> | Number of elements in the stack for this task. |

**60.9.3.5 #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.  |

## 60.9.4 Typedef Documentation

**60.9.4.1 typedef TaskHandle\_t task\_handler\_t**

**60.9.4.2 typedef portSTACK\_TYPE task\_stack\_t**

**60.9.4.3 typedef pdTASK\_CODE task\_t**

**60.9.4.4 typedef xSemaphoreHandle mutex\_t**

**60.9.4.5 typedef xSemaphoreHandle semaphore\_t**

**60.9.4.6 typedef EventBits\_t event\_flags\_t**

**60.9.4.7 typedef xQueueHandle msg\_queue\_t**



## Chapter 61

### Flexio\_spi\_driver

#### 61.1 Overview

##### Data Structures

- struct `flexio_spi_state_t`  
*Runtime state structure for FLEXIO SPI driver. [More...](#)*
- struct `flexio_spi_hwconfig_t`  
*FlexIO SPI hardware resource configuration. [More...](#)*
- struct `flexio_spi_userconfig_t`  
*User configuration structure for the FlexIO SPI driver. [More...](#)*

##### Typedefs

- typedef void(\* `flexio_spi_rx_callback_t`)(void \*spiState)  
*SPI receive callback function type.*

##### Enumerations

- enum `flexio_spi_status_t`  
*Error codes for the FLEXIO SPI driver.*
- enum `flexio_spi_master_slave_mode_t` {  
  `kFlexIOSpiMaster` = 1,  
  `kFlexIOSpiSlave` = 0 }  
*FlexIO SPI master or slave configuration.*
- enum `flexio_spi_shift_direction_t` {  
  `kFlexIOSpiMsbFirst` = 0,  
  `kFlexIOSpiLsbFirst` = 1 }  
*FlexIO SPI data shifter direction options.*
- enum `flexio_spi_clock_phase_t` {  
  `kFlexIOSpiClockPhase_FirstEdge` = 0,  
  `kFlexIOSpiClockPhase_SecondEdge` = 1 }  
*FlexIO SPI clock phase configuration.*
- enum `flexio_spi_data_bitcount_mode_t` {  
  `kFlexIOSpi8BitMode` = 8,  
  `kFlexIOSpi16BitMode` = 16 }  
*SPI data length mode options.*

##### FlexIO SPI Driver

- `flexio_spi_status_t FLEXIO_SPI_DRV_Init` (uint32\_t instance, `flexio_spi_state_t` \*spiState, `flexio_spi_userconfig_t` \*spiConfig)  
*Initializes a FlexIO-simulated SPI device.*
- void `FLEXIO_SPI_DRV_Deinit` (`flexio_spi_state_t` \*spiState)

## Overview

- Shuts down the FlexIO SPI.*

  - [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_SendDataBlocking](#) ([flexio\\_spi\\_state\\_t \\*spiState](#), [const uint8\\_t \\*txBuff](#), [uint32\\_t txSize](#), [uint32\\_t timeout](#))

*Sends (transmits) data out through the FlexIO-simulated SPI module using a blocking method.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_SendData](#) ([flexio\\_spi\\_state\\_t \\*spiState](#), [const uint8\\_t \\*txBuff](#), [uint32\\_t txSize](#))

*Sends (transmits) data through the FlexIO-simulated SPI module using a non-blocking method.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_GetTransmitStatus](#) ([flexio\\_spi\\_state\\_t \\*spiState](#), [uint32\\_t \\*bytesRemaining](#))

*Returns whether the previous FlexIO-simulated SPI transmit has finished.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_AbortSendingData](#) ([flexio\\_spi\\_state\\_t \\*spiState](#))

*Terminates a non-blocking FlexIO-simulated SPI transmission early.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_ReceiveDataBlocking](#) ([flexio\\_spi\\_state\\_t \\*spiState](#), [uint8\\_t \\*rxBuff](#), [uint32\\_t rxSize](#), [uint32\\_t timeout](#))

*Gets (receives) data from the FlexIO-simulated SPI module using a blocking method.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_ReceiveData](#) ([flexio\\_spi\\_state\\_t \\*spiState](#), [uint8\\_t \\*rxBuff](#), [uint32\\_t rxSize](#))

*Gets (receives) data from the FlexIO-simulated SPI module using a non-blocking method.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_GetReceiveStatus](#) ([flexio\\_spi\\_state\\_t \\*spiState](#), [uint32\\_t \\*bytesRemaining](#))

*Returns whether the previous FlexIO-simulated SPI receive is complete.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_AbortReceivingData](#) ([flexio\\_spi\\_state\\_t \\*spiState](#))

*Terminates a non-blocking FlexIO-simulated SPI receive early.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_TransferDataBlocking](#) ([flexio\\_spi\\_state\\_t \\*spiState](#), [const uint8\\_t \\*txBuff](#), [uint8\\_t \\*rxBuff](#), [uint32\\_t xSize](#), [uint32\\_t timeout](#))

*Transfers data through the FlexIO-simulated SPI module using a blocking method.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_TransferData](#) ([flexio\\_spi\\_state\\_t \\*spiState](#), [const uint8\\_t \\*txBuff](#), [uint8\\_t \\*rxBuff](#), [uint32\\_t xSize](#))

*Transfers data through the FlexIO-simulated SPI module using a non-blocking method.*
- [void FLEXIO\\_SPI\\_DRV\\_TX\\_IRQHandler](#) ([void \\*param](#))

*Interrupt handler for the FlexIO-simulated SPI Tx.*
- [void FLEXIO\\_SPI\\_DRV\\_RX\\_IRQHandler](#) ([void \\*param](#))

*Interrupt handler for the FlexIO-simulated SPI Rx.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_DmaSendDataBlocking](#) ([flexio\\_spi\\_state\\_t \\*spiState](#), [const uint8\\_t \\*txBuff](#), [uint32\\_t txSize](#), [uint32\\_t timeout](#))

*Sends (transmits) data out through the FlexIO-simulated SPI module using a DMA blocking method.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_DmaSendData](#) ([flexio\\_spi\\_state\\_t \\*spiState](#), [const uint8\\_t \\*txBuff](#), [uint32\\_t txSize](#))

*Sends (transmits) data through the FlexIO-simulated SPI module using a DMA non-blocking method.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_DmaGetTransmitStatus](#) ([flexio\\_spi\\_state\\_t \\*spiState](#), [uint32\\_t \\*bytesRemaining](#))

*Returns whether the previous FlexIO-simulated SPI-DMA transmit has finished.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_DmaAbortSendingData](#) ([flexio\\_spi\\_state\\_t \\*spiState](#))

*Terminates a non-blocking FlexIO-simulated SPI-DMA transmission early.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_DmaReceiveDataBlocking](#) ([flexio\\_spi\\_state\\_t \\*spiState](#), [uint8\\_t \\*rxBuff](#), [uint32\\_t rxSize](#), [uint32\\_t timeout](#))

*Gets (receives) data from the FlexIO-simulated SPI module using a DMA blocking method.*
- [flexio\\_spi\\_status\\_t FLEXIO\\_SPI\\_DRV\\_DmaReceiveData](#) ([flexio\\_spi\\_state\\_t \\*spiState](#), [uint8\\_t \\*rxBuff](#), [uint32\\_t rxSize](#))

- Gets (receives) data from the FlexIO-simulated SPI module using a DMA non-blocking method.*

  - `flexio_spi_status_t FLEXIO_SPI_DRV_DmaGetReceiveStatus` (`flexio_spi_state_t *spiState`, `uint32_t *bytesRemaining`)

*Returns whether the previous FlexIO-simulated SPI-DMA receive is complete.*

  - `flexio_spi_status_t FLEXIO_SPI_DRV_AbortDmaReceivingData` (`flexio_spi_state_t *spiState`)

*Terminates a non-blocking FlexIO-simulated SPI-DMA receive early.*

  - `flexio_spi_status_t FLEXIO_SPI_DRV_DmaTransferDataBlocking` (`flexio_spi_state_t *spiState`, `const uint8_t *txBuff`, `uint8_t *rxBuff`, `uint32_t xSize`, `uint32_t timeout`)

*Transfers data through the FlexIO-simulated SPI module using a DMA blocking method.*

  - `flexio_spi_status_t FLEXIO_SPI_DRV_DmaTransferData` (`flexio_spi_state_t *spiState`, `const uint8_t *txBuff`, `uint8_t *rxBuff`, `uint32_t xSize`)

*Transfers data through the FlexIO-simulated SPI module using a DMA non-blocking method.*

## 61.2 Data Structure Documentation

### 61.2.1 struct flexio\_spi\_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 `isXBusy`  
*True if there is an active transmit&receive simultaneously.*
- volatile bool `isTxBlocking`  
*True if transmit is blocking transaction.*
- volatile bool `isRxBlocking`  
*True if receive is blocking transaction.*
- volatile bool `isXBlocking`  
*True if transmit&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.*
- `semaphore_t xIrqSync`  
*Used to wait for ISR to complete its TX&RX business.*
- `flexio_spi_rx_callback_t rxCallback`  
*Callback to invoke after receiving byte.*
- void \* `rxCallbackParam`  
*Receive callback parameter pointer.*
- volatile bool `isTxUseDma`  
*True if Tx DMA channel has already been configured.*
- volatile bool `isRxUseDma`  
*True if Rx DMA channel has already been configured.*
- `dma_channel_t dmaSpiTx`  
*DMA Tx channel structure.*
- `dma_channel_t dmaSpiRx`  
*DMA Rx channel structure.*

## Data Structure Documentation

### 61.2.1.0.0.65 Field Documentation

- 61.2.1.0.0.65.1 `volatile bool flexio_spi_state_t::isTxBusy`
- 61.2.1.0.0.65.2 `volatile bool flexio_spi_state_t::isRxBusy`
- 61.2.1.0.0.65.3 `volatile bool flexio_spi_state_t::isXBusy`
- 61.2.1.0.0.65.4 `volatile bool flexio_spi_state_t::isTxBlocking`
- 61.2.1.0.0.65.5 `volatile bool flexio_spi_state_t::isRxBlocking`
- 61.2.1.0.0.65.6 `volatile bool flexio_spi_state_t::isXBlocking`
- 61.2.1.0.0.65.7 `semaphore_t flexio_spi_state_t::txIrqSync`
- 61.2.1.0.0.65.8 `semaphore_t flexio_spi_state_t::rxIrqSync`
- 61.2.1.0.0.65.9 `semaphore_t flexio_spi_state_t::xIrqSync`
- 61.2.1.0.0.65.10 `flexio_spi_rx_callback_t flexio_spi_state_t::rxCallback`
- 61.2.1.0.0.65.11 `void* flexio_spi_state_t::rxCallbackParam`
- 61.2.1.0.0.65.12 `volatile bool flexio_spi_state_t::isTxUseDma`
- 61.2.1.0.0.65.13 `volatile bool flexio_spi_state_t::isRxUseDma`
- 61.2.1.0.0.65.14 `dma_channel_t flexio_spi_state_t::dmaSpiRx`

### 61.2.2 struct flexio\_spi\_hwconfig\_t

These constants define the hardware resource used by FlexIO SPI master/slave device and includes the external pin and internal shifter and timer.

#### Data Fields

- `uint32_t sdoPinIdx`  
*Output pin index.*
- `uint32_t sdiPinIdx`  
*Input pin index.*
- `uint32_t sclkPinIdx`  
*Clock pin index.*
- `uint32_t csnPinIdx`  
*Chip select pin index.*
- `uint32_t shifterIdx [2]`  
*Select two shifters.*
- `uint32_t timerIdx [2]`  
*timer 0 is available for both master and slave.*

### 61.2.2.0.0.66 Field Documentation

#### 61.2.2.0.0.66.1 uint32\_t flexio\_spi\_hwconfig\_t::sclkPinIdx

Output for master, input for slave.

#### 61.2.2.0.0.66.2 uint32\_t flexio\_spi\_hwconfig\_t::csnPinIdx

Output for master, input for slave.

#### 61.2.2.0.0.66.3 uint32\_t flexio\_spi\_hwconfig\_t::timerIdx[2]

timer 1 would be only available for master and not used in slave mode.

### 61.2.3 struct flexio\_spi\_userconfig\_t

Use an instance of this structure with the [FLEXIO\\_SPI\\_DRV\\_Init\(\)](#) function. This enables configuration of the settings of the FlexIO SPI peripheral with a single function call. Settings include: SPI baud rate, data size, FlexIO SPI mode and FlexIO hardware resource resource.

#### Data Fields

- [flexio\\_spi\\_master\\_slave\\_mode\\_t spiMode](#)  
*Selects Master or Slave mode.*
- uint32\_t [baudRate](#)  
*Baudrate configuration.*
- [flexio\\_spi\\_clock\\_phase\\_t clkPhase](#)  
*Clock phase configuration.*
- [flexio\\_spi\\_data\\_bitcount\\_mode\\_t dataSize](#)  
*SPI data length mode.*
- [flexio\\_spi\\_shift\\_direction\\_t bitDirection](#)  
*SPI data shifter direction options.*
- [flexio\\_spi\\_hwconfig\\_t spiHwConfig](#)  
*FlexIO SPI Resource configuration.*

### 61.3 Enumeration Type Documentation

#### 61.3.1 enum flexio\_spi\_status\_t

#### 61.3.2 enum flexio\_spi\_master\_slave\_mode\_t

Enumerator

- kFlexIOSpiMaster* SPI peripheral operates in master mode.
- kFlexIOSpiSlave* SPI peripheral operates in slave mode.

## Function Documentation

### 61.3.3 enum flexio\_spi\_shift\_direction\_t

Enumerator

*kFlexIOSpiMsbFirst* Data transfers start with most significant bit.

*kFlexIOSpiLsbFirst* Data transfers start with least significant bit.

### 61.3.4 enum flexio\_spi\_clock\_phase\_t

Enumerator

*kFlexIOSpiClockPhase\_FirstEdge* First edge on SPSCCK occurs at the middle of the first cycle of a data transfer.

*kFlexIOSpiClockPhase\_SecondEdge* First edge on SPSCCK occurs at the start of the first cycle of a data transfer.

### 61.3.5 enum flexio\_spi\_data\_bitcount\_mode\_t

Enumerator

*kFlexIOSpi8BitMode* 8-bit data transmission mode

*kFlexIOSpi16BitMode* 16-bit data transmission mode

## 61.4 Function Documentation

### 61.4.1 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_Init ( uint32\_t instance, flexio\_spi\_state\_t \* spiState, flexio\_spi\_userconfig\_t \* spiConfig )

This function initializes the run-time state structure to keep track of the on-going transfers and the module to user defined settings and default settings. It also configures the underlying FlexIO pin, shifter, and timer. This is an example to set up the [flexio\\_spi\\_state\\_t](#) and the [flexio\\_spi\\_userconfig\\_t](#) parameters and to call the FLEXIO\_SPI\_DRV\_Init function

```
flexio_spi_userconif_t spiConfig;
spiConfig.spiMode = kFlexIOSpiMaster;
spiConfig.baudRate = 100000;
spiConfig.clkPhase = kFlexIOSpiClockPhase_FirstEdge;
spiConfig.dataSize = kFlexIOSpi8BitMode;
spiConfig.spiHwConfig.sdoPinIdx = 0;
spiConfig.spiHwConfig.sdiPinIdx = 1;
spiConfig.spiHwConfig.sclkPinIdx = 2;
spiConfig.spiHwConfig.csnPinIdx = 3;
spiConfig.spiHwConfig.shifterIdx = {0,1};
spiConfig.spiHwConfig.timerIdx = {0,1};
```



## Parameters

|                  |                                                                                                                                                                                                                                                               |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>  | The FlexIO instance number.                                                                                                                                                                                                                                   |
| <i>spiState</i>  | A pointer to the global FlexIO SPI driver state structure memory. The user passes in the memory for the run-time state structure. The FlexIO SPI driver populates the members. This run-time state structure keeps track of the current transfer in progress. |
| <i>spiConfig</i> | The user configuration structure of type <a href="#">flexio_spi_userconfig_t</a> . The user populates the members of this structure and passes the pointer of this structure to this function.                                                                |

## Returns

An error code or `kStatus_FlexIO_SPI_Success`.

### 61.4.2 void FLEXIO\_SPI\_DRV\_Deinit ( flexio\_spi\_state\_t \* spiState )

This function disables the FlexIO-simulated SPI trigger.

## Parameters

|                 |                                                 |
|-----------------|-------------------------------------------------|
| <i>spiState</i> | The run-time structure of FLEXIO simulated SPI. |
|-----------------|-------------------------------------------------|

### 61.4.3 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_SendDataBlocking ( flexio\_spi\_state\_t \* spiState, const uint8\_t \* txBuff, uint32\_t txSize, uint32\_t timeout )

## Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI.                         |
| <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_FlexIO_SPI_Success`.

### 61.4.4 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_SendData ( flexio\_spi\_state\_t \* spiState, const uint8\_t \* txBuff, uint32\_t txSize )

## Function Documentation

### Parameters

|                 |                                                                     |
|-----------------|---------------------------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI.                     |
| <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_FlexIO_SPI_Success`.

### 61.4.5 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_GetTransmitStatus ( flexio\_spi\_state\_t \* spiState, uint32\_t \* bytesRemaining )

### Parameters

|                        |                                                                                                            |
|------------------------|------------------------------------------------------------------------------------------------------------|
| <i>spiState</i>        | The run-time structure of the FlexIO-simulated SPI.                                                        |
| <i>bytes-Remaining</i> | A pointer to a value that is populated with the number of bytes that are remaining in the active transfer. |

### Returns

An error code or `kStatus_FlexIO_SPI_Success`.

### Return values

|                                   |                                                                                                                                       |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_FlexIO_SPI-Success</i> | The transmit has completed successfully.                                                                                              |
| <i>kStatus_FlexIO_SPI-Tx-Busy</i> | The transmit is still in progress. <i>bytesTransmitted</i> is filled with the number of bytes which are transmitted up to that point. |

### 61.4.6 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_AbortSendingData ( flexio\_spi\_state\_t \* spiState )

### Parameters

---

|                 |                                                 |
|-----------------|-------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI. |
|-----------------|-------------------------------------------------|

Returns

An error code or `kStatus_FlexIO_SPI_Success`.

Return values

|                                                |                                           |
|------------------------------------------------|-------------------------------------------|
| <i>kStatus_FlexIO_SPI_Success</i>              | The transmit was successful.              |
| <i>kStatus_FlexIO_SPI_NoTransmitInProgress</i> | No transmission is currently in progress. |

#### 61.4.7 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_ReceiveDataBlocking ( flexio\_spi\_state\_t \* spiState, uint8\_t \* rxBuff, uint32\_t rxSize, uint32\_t timeout )

Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI.                         |
| <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_FlexIO_SPI_Success`.

#### 61.4.8 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_ReceiveData ( flexio\_spi\_state\_t \* spiState, uint8\_t \* rxBuff, uint32\_t rxSize )

Parameters

## Function Documentation

|                 |                                                                    |
|-----------------|--------------------------------------------------------------------|
| <i>spiState</i> | The run-time structure of the FlexIO-simulated SPI.                |
| <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_FlexIO_SPI_Success`.

### 61.4.9 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_GetReceiveStatus ( flexio\_spi\_state\_t \* spiState, uint32\_t \* bytesRemaining )

#### Parameters

|                        |                                                                                                                         |
|------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>spiState</i>        | The run-time structure of FlexIO-simulated SPI.                                                                         |
| <i>bytes-Remaining</i> | A pointer to a value that is populated with the number of bytes which still need to be received in the active transfer. |

### Returns

An error code or `kStatus_FlexIO_SPI_Success`.

#### Return values

|                                   |                                                                                                                                |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_FlexIO_SPI_Success</i> | The receive has completed successfully.                                                                                        |
| <i>kStatus_FlexIO_SPI_Rx-Busy</i> | The receive is still in progress. <i>bytesReceived</i> is filled with the number of bytes which are received up to that point. |

### 61.4.10 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_AbortReceivingData ( flexio\_spi\_state\_t \* spiState )

#### Parameters

|                 |                                                 |
|-----------------|-------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI. |
|-----------------|-------------------------------------------------|

Returns

An error code or `kStatus_SPI_Success`.

Return values

|                                                |                                      |
|------------------------------------------------|--------------------------------------|
| <i>kStatus_FlexIO_SPI_Success</i>              | The receive was successful.          |
| <i>kStatus_FlexIO_SPI_NoTransmitInProgress</i> | No receive is currently in progress. |

**61.4.11 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_TransferDataBlocking ( flexio\_spi\_state\_t \* spiState, const uint8\_t \* txBuff, uint8\_t \* rxBuff, uint32\_t xSize, uint32\_t timeout )**

Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI.                         |
| <i>txBuff</i>   | A pointer to the source buffer containing 8-bit data chars to send.     |
| <i>rxBuff</i>   | A pointer to the buffer containing 8-bit read data chars received.      |
| <i>xSize</i>    | The number of bytes to send&receive.                                    |
| <i>timeout</i>  | A timeout value for RTOS abstraction sync control in milliseconds (ms). |

Returns

An error code or `kStatus_FlexIO_SPI_Success`.

**61.4.12 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_TransferData ( flexio\_spi\_state\_t \* spiState, const uint8\_t \* txBuff, uint8\_t \* rxBuff, uint32\_t xSize )**

Parameters

---

## Function Documentation

|                 |                                                                     |
|-----------------|---------------------------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI.                     |
| <i>txBuff</i>   | A pointer to the source buffer containing 8-bit data chars to send. |
| <i>rxBuff</i>   | A pointer to the buffer containing 8-bit read data chars received.  |
| <i>xSize</i>    | The number of bytes to send.                                        |

Returns

An error code or `kStatus_FlexIO_SPI_Success`.

### 61.4.13 void FLEXIO\_SPI\_DRV\_TX\_IRQHandler ( void \* *param* )

Parameters

|              |                                                 |
|--------------|-------------------------------------------------|
| <i>param</i> | The run-time structure of FlexIO simulated SPI. |
|--------------|-------------------------------------------------|

### 61.4.14 void FLEXIO\_SPI\_DRV\_RX\_IRQHandler ( void \* *param* )

Parameters

|              |                                                 |
|--------------|-------------------------------------------------|
| <i>param</i> | The run-time structure of FLEXIO simulated SPI. |
|--------------|-------------------------------------------------|

### 61.4.15 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_DmaSendDataBlocking ( flexio\_spi\_state\_t \* *spiState*, const uint8\_t \* *txBuff*, uint32\_t *txSize*, uint32\_t *timeout* )

Parameters

|                 |                                                                          |
|-----------------|--------------------------------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI.                          |
| <i>txBuff</i>   | A pointer to the source buffer containing 8-bit data characters 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_FlexIO_SPI_Success`.

**61.4.16** `flexio_spi_status_t FLEXIO_SPI_DRV_DmaSendData ( flexio_spi_state_t * spiState, const uint8_t * txBuff, uint32_t txSize )`

## Function Documentation

### Parameters

|                 |                                                                     |
|-----------------|---------------------------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI.                     |
| <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_FlexIO_SPI_Success`.

### 61.4.17 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_DmaGetTransmitStatus ( flexio\_spi\_state\_t \* *spiState*, uint32\_t \* *bytesRemaining* )

### Parameters

|                        |                                                                                                            |
|------------------------|------------------------------------------------------------------------------------------------------------|
| <i>spiState</i>        | The run-time structure of FlexIO-simulated SPI.                                                            |
| <i>bytes-Remaining</i> | A pointer to a value that is populated with the number of bytes that are remaining in the active transfer. |

### Returns

An error code or `kStatus_FlexIO_SPI_Success`.

### Return values

|                                   |                                                                                                                                       |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_FlexIO_SPI_Success</i> | The transmit has completed successfully.                                                                                              |
| <i>kStatus_FlexIO_SPI_Tx-Busy</i> | The transmit is still in progress. <i>bytesTransmitted</i> is filled with the number of bytes which are transmitted up to that point. |

### 61.4.18 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_DmaAbortSendingData ( flexio\_spi\_state\_t \* *spiState* )

### Parameters

---



|                 |                                                 |
|-----------------|-------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI. |
|-----------------|-------------------------------------------------|

Returns

An error code or `kStatus_FlexIO_SPI_Success`.

Return values

|                                                |                                           |
|------------------------------------------------|-------------------------------------------|
| <i>kStatus_FlexIO_SPI_Success</i>              | The transmit was successful.              |
| <i>kStatus_FlexIO_SPI_NoTransmitInProgress</i> | No transmission is currently in progress. |

#### 61.4.19 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_DmaReceiveDataBlocking ( flexio\_spi\_state\_t \* spiState, uint8\_t \* rxBuff, uint32\_t rxSize, uint32\_t timeout )

Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI.                         |
| <i>rxBuff</i>   | A pointer to the buffer containing 8-bit read data characters 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_FlexIO_SPI_Success`.

#### 61.4.20 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_DmaReceiveData ( flexio\_spi\_state\_t \* spiState, uint8\_t \* rxBuff, uint32\_t rxSize )

Parameters

|                 |                                                                         |
|-----------------|-------------------------------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI.                         |
| <i>rxBuff</i>   | A pointer to the buffer containing 8-bit read data characters received. |
| <i>rxSize</i>   | The number of bytes to receive.                                         |

## Function Documentation

Returns

An error code or `kStatus_FlexIO_SPI_Success`.

### 61.4.21 `flexio_spi_status_t` FLEXIO\_SPI\_DRV\_DmaGetReceiveStatus ( `flexio_spi_state_t * spiState`, `uint32_t * bytesRemaining` )

Parameters

|                        |                                                                                                                         |
|------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>spiState</i>        | The run-time structure of FlexIO-simulated SPI.                                                                         |
| <i>bytes-Remaining</i> | A pointer to a value that is populated with the number of bytes which still need to be received in the active transfer. |

Returns

An error code or `kStatus_FlexIO_SPI_Success`.

Return values

|                                   |                                                                                                                                |
|-----------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_FlexIO_SPI_Success</i> | The receive has completed successfully.                                                                                        |
| <i>kStatus_FlexIO_SPI_Rx-Busy</i> | The receive is still in progress. <i>bytesReceived</i> is filled with the number of bytes which are received up to that point. |

### 61.4.22 `flexio_spi_status_t` FLEXIO\_SPI\_DRV\_AbortDmaReceivingData ( `flexio_spi_state_t * spiState` )

Parameters

|                 |                                                 |
|-----------------|-------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI. |
|-----------------|-------------------------------------------------|

Returns

An error code or `kStatus_SPI_Success`.

Return values

|                                                |                                      |
|------------------------------------------------|--------------------------------------|
| <i>kStatus_FlexIO_SPI_Success</i>              | The receive was successful.          |
| <i>kStatus_FlexIO_SPI_NoTransmitInProgress</i> | No receive is currently in progress. |

**61.4.23 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_DmaTransferDataBlocking ( flexio\_spi\_state\_t \* spiState, const uint8\_t \* txBuff, uint8\_t \* rxBuff, uint32\_t xSize, uint32\_t timeout )**

Parameters

|                 |                                                                          |
|-----------------|--------------------------------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI.                          |
| <i>txBuff</i>   | A pointer to the source buffer containing 8-bit data characters to send. |
| <i>rxBuff</i>   | A pointer to the buffer containing 8-bit read data chars received.       |
| <i>xSize</i>    | The number of bytes to send&receive.                                     |
| <i>timeout</i>  | A timeout value for RTOS abstraction sync control in milliseconds (ms).  |

Returns

An error code or kStatus\_FlexIO\_SPI\_Success.

**61.4.24 flexio\_spi\_status\_t FLEXIO\_SPI\_DRV\_DmaTransferData ( flexio\_spi\_state\_t \* spiState, const uint8\_t \* txBuff, uint8\_t \* rxBuff, uint32\_t xSize )**

Parameters

|                 |                                                                          |
|-----------------|--------------------------------------------------------------------------|
| <i>spiState</i> | The run-time structure of FlexIO-simulated SPI.                          |
| <i>txBuff</i>   | A pointer to the source buffer containing 8-bit data characters to send. |
| <i>rxBuff</i>   | A pointer to the buffer containing 8-bit read data chars received.       |
| <i>xSize</i>    | The number of bytes to send.                                             |

Returns

An error code or kStatus\_FlexIO\_SPI\_Success.



---

## Function Documentation

## Chapter 62

### Flexio\_uart\_driver

#### 62.1 Overview

##### Data Structures

- struct `flexio_uart_dmastate_t`  
*Runtime state structure for FlexIO UART driver with DMA. [More...](#)*
- struct `flexio_uartdma_userconfig_t`  
*User configuration structure for the FlexIO UART driver with DMA. [More...](#)*
- struct `flexio_uartdma_userconfig_t`  
*User configuration structure for the FlexIO UART driver. [More...](#)*
- struct `flexio_uart_edmastate_t`  
*Runtime state of the FlexIO UART driver. [More...](#)*

##### FlexIO UART DMA Driver

- `flexio_uart_status_t FLEXIO_UART_DRV_DmaInit` (`uint32_t` instance, `flexio_uart_dmastate_t *uartDmaState`, `const flexio_uartdma_userconfig_t *uartDmaConfig`)  
*Initializes a FlexIO-simulated UART device to work with DMA.*
- `void FLEXIO_UART_DRV_DmaDeinit` (`flexio_uart_dmastate_t *uartDmaState`)  
*Shuts down the FlexIO UART.*
- `flexio_uart_status_t FLEXIO_UART_DRV_DmaSendDataBlocking` (`flexio_uart_dmastate_t *uartDmaState`, `const uint8_t *txBuff`, `uint32_t txSize`, `uint32_t timeout`)  
*Sends (transmits) data out through the FlexIO-simulated UART-DMA module using a blocking method.*
- `flexio_uart_status_t FLEXIO_UART_DRV_DmaSendData` (`flexio_uart_dmastate_t *uartDmaState`, `const uint8_t *txBuff`, `uint32_t txSize`)  
*Sends (transmits) data through the FlexIO-simulated UART-DMA module using a non-blocking method.*
- `flexio_uart_status_t FLEXIO_UART_DRV_DmaGetTransmitStatus` (`flexio_uart_dmastate_t *uartDmaState`, `uint32_t *bytesRemaining`)  
*Returns whether the previous FlexIO-simulated UART-DMA transmit has finished.*
- `flexio_uart_status_t FLEXIO_UART_DRV_DmaAbortSendingData` (`flexio_uart_dmastate_t *uartDmaState`)  
*Terminates a non-blocking FlexIO-simulated UART-DMA transmission early.*
- `flexio_uart_status_t FLEXIO_UART_DRV_DmaReceiveDataBlocking` (`flexio_uart_dmastate_t *uartDmaState`, `uint8_t *rxBuff`, `uint32_t rxSize`, `uint32_t timeout`)  
*Gets (receives) data from the FlexIO-simulated UART-DMA module using a blocking method.*
- `flexio_uart_status_t FLEXIO_UART_DRV_DmaReceiveData` (`flexio_uart_dmastate_t *uartDmaState`, `uint8_t *rxBuff`, `uint32_t rxSize`)  
*Gets (receives) data from the FlexIO-simulated UART-DMA module using a non-blocking method.*
- `flexio_uart_status_t FLEXIO_UART_DRV_DmaGetReceiveStatus` (`flexio_uart_dmastate_t *uartDmaState`, `uint32_t *bytesRemaining`)  
*Returns whether the previous FlexIO-simulated UART-DMA receive is complete.*
- `flexio_uart_status_t FLEXIO_UART_DRV_DmaAbortReceivingData` (`flexio_uart_dmastate_t *uartDmaState`)

## Data Structure Documentation

*Terminates a non-blocking FlexIO-simulated UART-DMA receive early.*

### FlexIO UART eDMA Driver

- flexio\_uart\_status\_t [FLEXIO\\_UART\\_DRV\\_EdmaInit](#) (uint32\_t instance, flexio\_uart\_edmastate\_t \*uartEdmaState, const flexio\_uartedma\_userconfig\_t \*uartEdmaConfig)  
*Initializes a FlexIO-simulated UART device to work with eDMA.*
- void [FLEXIO\\_UART\\_DRV\\_EdmaDeinit](#) (flexio\_uart\_edmastate\_t \*uartEdmaState)  
*Shuts down the FlexIO UART.*
- flexio\_uart\_status\_t [FLEXIO\\_UART\\_DRV\\_EdmaSendDataBlocking](#) (flexio\_uart\_edmastate\_t \*uartEdmaState, const uint8\_t \*txBuff, uint32\_t txSize, uint32\_t timeout)  
*Sends (transmits) data out through the FlexIO-simulated UART-EDMA module using a blocking method.*
- flexio\_uart\_status\_t [FLEXIO\\_UART\\_DRV\\_EdmaSendData](#) (flexio\_uart\_edmastate\_t \*uartEdmaState, const uint8\_t \*txBuff, uint32\_t txSize)  
*Sends (transmits) data through the FlexIO-simulated UART-EDMA module using a non-blocking method.*
- flexio\_uart\_status\_t [FLEXIO\\_UART\\_DRV\\_EdmaGetTransmitStatus](#) (flexio\_uart\_edmastate\_t \*uartEdmaState, uint32\_t \*bytesRemaining)  
*Returns whether the previous FlexIO-simulated UART-EDMA transmit has finished.*
- flexio\_uart\_status\_t [FLEXIO\\_UART\\_DRV\\_EdmaAbortSendingData](#) (flexio\_uart\_edmastate\_t \*uartEdmaState)  
*Terminates a non-blocking FlexIO-simulated UART-eDMA transmission early.*
- flexio\_uart\_status\_t [FLEXIO\\_UART\\_DRV\\_EdmaReceiveDataBlocking](#) (flexio\_uart\_edmastate\_t \*uartEdmaState, uint8\_t \*rxBuff, uint32\_t rxSize, uint32\_t timeout)  
*Gets (receives) data from the FlexIO-simulated UART-eDMA module using a blocking method.*
- flexio\_uart\_status\_t [FLEXIO\\_UART\\_DRV\\_EdmaReceiveData](#) (flexio\_uart\_edmastate\_t \*uartEdmaState, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Gets (receives) data from the FlexIO-simulated UART-eDMA module using a non-blocking method.*
- flexio\_uart\_status\_t [FLEXIO\\_UART\\_DRV\\_EdmaGetReceiveStatus](#) (flexio\_uart\_edmastate\_t \*uartEdmaState, uint32\_t \*bytesRemaining)  
*Returns whether the previous FlexIO-simulated UART-eDMA receive is complete.*
- flexio\_uart\_status\_t [FLEXIO\\_UART\\_DRV\\_EdmaAbortReceivingData](#) (flexio\_uart\_edmastate\_t \*uartEdmaState)  
*Terminates a non-blocking FlexIO-simulated UART-eDMA receive early.*

## 62.2 Data Structure Documentation

### 62.2.1 struct flexio\_uart\_dmastate\_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.
  - [dma\\_channel\\_t dmaUartTx](#)
- DMA channel used for send.
  - [dma\\_channel\\_t dmaUartRx](#)
- DMA channel used for receive.
  - [dma\\_channel\\_t dmaUartRx](#)

### 62.2.1.0.0.67 Field Documentation

62.2.1.0.0.67.1 volatile bool flexio\_uart\_dmastate\_t::isTxBusy

62.2.1.0.0.67.2 volatile bool flexio\_uart\_dmastate\_t::isRxBusy

62.2.1.0.0.67.3 volatile bool flexio\_uart\_dmastate\_t::isTxBlocking

62.2.1.0.0.67.4 volatile bool flexio\_uart\_dmastate\_t::isRxBlocking

62.2.1.0.0.67.5 semaphore\_t flexio\_uart\_dmastate\_t::txIrqSync

62.2.1.0.0.67.6 semaphore\_t flexio\_uart\_dmastate\_t::rxIrqSync

62.2.1.0.0.67.7 dma\_channel\_t flexio\_uart\_dmastate\_t::dmaUartTx

62.2.1.0.0.67.8 dma\_channel\_t flexio\_uart\_dmastate\_t::dmaUartRx

### 62.2.2 struct flexio\_uartdma\_userconfig\_t

Use an instance of this structure with the [FLEXIO\\_UART\\_DRV\\_DmaInit\(\)](#) 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.
- flexio\_uart\_bit\_count\_per\_char\_t [bitCountPerChar](#)  
 number of bits, 5/6/7/8 bits configurable
- flexio\_uart\_mode\_t [uartMode](#)  
 FlexIO UART working modes: Tx only, Rx only, or both.
- flexio\_uart\_hwconfig\_t [txConfig](#)  
 FlexIO UART TX device hardware resource configuration.
- flexio\_uart\_hwconfig\_t [rxConfig](#)  
 FlexIO UART RX device hardware resource configuration.

## Data Structure Documentation

### 62.2.3 struct flexio\_uartdma\_userconfig\_t

Use an instance of this structure with the `FLEXIO_UART_DRV_Init()` function. This enables configuration of the settings of the FlexIO UART peripheral with a single function call. Settings include: UART baud rate, the number of bits per data word, FlexIO UART mode, TX hardware resource and Rx hardware resource.

### 62.2.4 struct flexio\_uart\_edmastate\_t

This structure holds data that are used by the FlexIO 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 FlexIO UART driver fills out the members.

#### Data Fields

- 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.*

#### 62.2.4.0.0.68 Field Documentation

**62.2.4.0.0.68.1** volatile bool flexio\_uart\_edmastate\_t::isTxBlocking

**62.2.4.0.0.68.2** volatile bool flexio\_uart\_edmastate\_t::isRxBlocking

**62.2.4.0.0.68.3** semaphore\_t flexio\_uart\_edmastate\_t::txIrqSync

**62.2.4.0.0.68.4** semaphore\_t flexio\_uart\_edmastate\_t::rxIrqSync



## 62.3 Function Documentation

### 62.3.1 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_DmaInit ( uint32\_t instance, flexio\_uart\_dmastate\_t \* uartDmaState, const flexio\_uartdma\_userconfig\_t \* uartDmaConfig )

This function initializes the run-time state structure to keep track of the on-going transfers and the module to user-defined settings and default settings. It also configures the underlying FlexIO pin, shifter, and timer resource, and enables the FlexIO simulated UART module DMA interrupt. This example shows how to set up the flexio\_uartdma\_state\_t and the flexio\_uartdma\_userconfig\_t parameters and how to call the FLEXIO\_UART\_DRV\_DmaInit function by passing in these parameters:

```
flexio_uartdma_userconfig_t uartDmaConfig;
uartDmaConfig.baudRate = 9600;
uartDmaConfig.bitCountPerChar = kUart8BitsPerChar;
uartDmaConfig.uartMode = flexioUART_TxRx;
```

#### Parameters

|                      |                                                                                                                                                                                                                                                                 |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>      | The FlexIO instance number.                                                                                                                                                                                                                                     |
| <i>uartDmaState</i>  | A pointer to the global FlexIO UART driver state structure memory. The user passes in the memory for the run-time state structure. The FlexIO UART driver populates the members. This run-time state structure keeps track of the current transfer in progress. |
| <i>uartDmaConfig</i> | The user configuration structure of type flexio_uartdma_userconfig_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\_FlexIO\_UART\_Success.

### 62.3.2 void FLEXIO\_UART\_DRV\_DmaDeinit ( flexio\_uart\_dmastate\_t \* uartDmaState )

This function disables the FlexIO-simulated UART-DMA trigger.

#### Parameters

---

## Function Documentation

|                     |                                                  |
|---------------------|--------------------------------------------------|
| <i>uartDmaState</i> | The run-time structure of FlexIO-simulated UART. |
|---------------------|--------------------------------------------------|

**62.3.3 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_DmaSendDataBlocking ( flexio\_uart\_dmastate\_t \* *uartDmaState*, const uint8\_t \* *txBuff*, uint32\_t *txSize*, uint32\_t *timeout* )**

### Parameters

|                     |                                                                         |
|---------------------|-------------------------------------------------------------------------|
| <i>uartDmaState</i> | The run-time structure of FlexIO-simulated UART.                        |
| <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\_FlexIO\_UART\_Success.

**62.3.4 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_DmaSendData ( flexio\_uart\_dmastate\_t \* *uartDmaState*, const uint8\_t \* *txBuff*, uint32\_t *txSize* )**

### Parameters

|                     |                                                                     |
|---------------------|---------------------------------------------------------------------|
| <i>uartDmaState</i> | The run-time structure of FlexIO-simulated UART.                    |
| <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\_FlexIO\_UART\_Success.

**62.3.5 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_DmaGetTransmitStatus ( flexio\_uart\_dmastate\_t \* *uartDmaState*, uint32\_t \* *bytesRemaining* )**

Parameters

|                        |                                                                                                            |
|------------------------|------------------------------------------------------------------------------------------------------------|
| <i>uartDmaState</i>    | The run-time structure of FlexIO-simulated UART.                                                           |
| <i>bytes-Remaining</i> | A pointer to a value that is populated with the number of bytes that are remaining in the active transfer. |

Returns

An error code or `kStatus_FlexIO_UART_Success`.

Return values

|                                    |                                                                                                                                       |
|------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_FlexIO_UART_Success</i> | The transmit has completed successfully.                                                                                              |
| <i>kStatus_FlexIO_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. |

**62.3.6 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_DmaAbortSendingData ( flexio\_uart\_dmastate\_t \* *uartDmaState* )**

Parameters

|                     |                                                  |
|---------------------|--------------------------------------------------|
| <i>uartDmaState</i> | The run-time structure of FlexIO-simulated UART. |
|---------------------|--------------------------------------------------|

Returns

An error code or `kStatus_FlexIO_UART_Success`.

Return values

|                                                 |                                           |
|-------------------------------------------------|-------------------------------------------|
| <i>kStatus_FlexIO_UART_Success</i>              | The transmit was successful.              |
| <i>kStatus_FlexIO_UART_NoTransmitInProgress</i> | No transmission is currently in progress. |

**62.3.7 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_DmaReceiveDataBlocking ( flexio\_uart\_dmastate\_t \* *uartDmaState*, uint8\_t \* *rxBuff*, uint32\_t *rxSize*, uint32\_t *timeout* )**

## Function Documentation

### Parameters

|                     |                                                                         |
|---------------------|-------------------------------------------------------------------------|
| <i>uartDmaState</i> | The run-time structure of FlexIO-simulated UART.                        |
| <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_FlexIO_UART_Success`.

### 62.3.8 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_DmaReceiveData ( flexio\_uart\_dmastate\_t \* *uartDmaState*, uint8\_t \* *rxBuff*, uint32\_t *rxSize* )

### Parameters

|                     |                                                                    |
|---------------------|--------------------------------------------------------------------|
| <i>uartDmaState</i> | The run-time structure of FlexIO-simulated UART.                   |
| <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_FlexIO_UART_Success`.

### 62.3.9 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_DmaGetReceiveStatus ( flexio\_uart\_dmastate\_t \* *uartDmaState*, uint32\_t \* *bytesRemaining* )

### Parameters

|                        |                                                                                                                         |
|------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>uartDmaState</i>    | The run-time structure of FlexIO-simulated UART.                                                                        |
| <i>bytes-Remaining</i> | A pointer to a value that is populated with the number of bytes which still need to be received in the active transfer. |

### Returns

An error code or `kStatus_FlexIO_UART_Success`.

Return values

|                                    |                                                                                                                                |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_FlexIO_UART_Success</i> | The receive has completed successfully.                                                                                        |
| <i>kStatus_FlexIO_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. |

**62.3.10 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_DmaAbortReceivingData ( flexio\_uart\_dmastate\_t \* *uartDmaState* )**

Parameters

|                     |                                                  |
|---------------------|--------------------------------------------------|
| <i>uartDmaState</i> | The run-time structure of FlexIO-simulated UART. |
|---------------------|--------------------------------------------------|

Returns

An error code or *kStatus\_UART\_Success*.

Return values

|                                                 |                                      |
|-------------------------------------------------|--------------------------------------|
| <i>kStatus_FlexIO_UART_Success</i>              | The receive was successful.          |
| <i>kStatus_FlexIO_UART_NoTransmitInProgress</i> | No receive is currently in progress. |

**62.3.11 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_EdmaInit ( uint32\_t *instance*, flexio\_uart\_edmastate\_t \* *uartEdmaState*, const flexio\_uartedma\_userconfig\_t \* *uartEdmaConfig* )**

This function initializes the run-time state structure to keep track of the on-going transfers, initializes the module to user-defined settings and default settings, configures underlying FlexIO pin, shifter, and timer resource, and enables the FlexIO simulated UART module eDMA interrupt. This example shows how to set up the *flexio\_uartedma\_state\_t* and the *flexio\_uartedma\_userconfig\_t* parameters and how to call the *FLEXIO\_UART\_DRV\_EdmaInit* function by passing in these parameters:

```
flexio_uartedma_userconfig_t uartEdmaConfig;
uartEdmaConfig.baudRate = 9600;
uartEdmaConfig.bitCountPerChar = kUart8BitsPerChar;
uartEdmaConfig.uartMode = flexioUART_TxRx;
```

## Function Documentation

### Parameters

|                        |                                                                                                                                                                                                                                                                 |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>        | The FlexIO instance number.                                                                                                                                                                                                                                     |
| <i>uartEdmaState</i>   | A pointer to the global FlexIO UART driver state structure memory. The user passes in the memory for the run-time state structure. The FlexIO UART driver populates the members. This run-time state structure keeps track of the current transfer in progress. |
| <i>uartEdma-Config</i> | The user configuration structure of type <a href="#">flexio_uartedma_userconfig_t</a> . The user populates the members of this structure and passes the pointer of this structure to this function.                                                             |

### Returns

An error code or `kStatus_FlexIO_UART_Success`.

### 62.3.12 void FLEXIO\_UART\_DRV\_EdmaDeinit ( flexio\_uart\_edmastate\_t \* *uartEdmaState* )

This function disables the FlexIO-simulated UART-eDMA trigger.

### Parameters

|                      |                                                  |
|----------------------|--------------------------------------------------|
| <i>uartEdmaState</i> | The run-time structure of FlexIO-simulated UART. |
|----------------------|--------------------------------------------------|

### 62.3.13 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_EdmaSendDataBlocking ( flexio\_uart\_edmastate\_t \* *uartEdmaState*, const uint8\_t \* *txBuff*, uint32\_t *txSize*, uint32\_t *timeout* )

### Parameters

|                      |                                                                         |
|----------------------|-------------------------------------------------------------------------|
| <i>uartEdmaState</i> | The run-time structure of FlexIO-simulated UART.                        |
| <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_FlexIO_UART_Success`.

**62.3.14 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_EdmaSendData ( flexio\_uart\_edmastate\_t \* *uartEdmaState*, const uint8\_t \* *txBuff*, uint32\_t *txSize* )**

## Function Documentation

### Parameters

|                      |                                                                     |
|----------------------|---------------------------------------------------------------------|
| <i>uartEdmaState</i> | The run-time structure of FlexIO-simulated UART.                    |
| <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_FlexIO_UART_Success`.

### 62.3.15 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_EdmaGetTransmitStatus ( flexio\_uart\_edmastate\_t \* *uartEdmaState*, uint32\_t \* *bytesRemaining* )

### Parameters

|                        |                                                                                                            |
|------------------------|------------------------------------------------------------------------------------------------------------|
| <i>uartEdmaState</i>   | The run-time structure of FlexIO-simulated UART.                                                           |
| <i>bytes-Remaining</i> | A pointer to a value that is populated with the number of bytes that are remaining in the active transfer. |

### Returns

An error code or `kStatus_FlexIO_UART_Success`.

### Return values

|                                     |                                                                                                                                       |
|-------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_FlexIO_UART_-Success</i> | The transmit has completed successfully.                                                                                              |
| <i>kStatus_FlexIO_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. |

### 62.3.16 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_EdmaAbortSendingData ( flexio\_uart\_edmastate\_t \* *uartEdmaState* )

### Parameters

---



|                      |                                                  |
|----------------------|--------------------------------------------------|
| <i>uartEdmaState</i> | The run-time structure of FlexIO-simulated UART. |
|----------------------|--------------------------------------------------|

Returns

An error code or `kStatus_FlexIO_UART_Success`.

Return values

|                                                 |                                           |
|-------------------------------------------------|-------------------------------------------|
| <i>kStatus_FlexIO_UART_Success</i>              | The transmit was successful.              |
| <i>kStatus_FlexIO_UART_NoTransmitInProgress</i> | No transmission is currently in progress. |

**62.3.17 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_EdmaReceiveDataBlocking ( flexio\_uart\_edmastate\_t \* *uartEdmaState*, uint8\_t \* *rxBuff*, uint32\_t *rxSize*, uint32\_t *timeout* )**

Parameters

|                      |                                                                         |
|----------------------|-------------------------------------------------------------------------|
| <i>uartEdmaState</i> | The run-time structure of FlexIO-simulated UART.                        |
| <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_FlexIO_UART_Success`.

**62.3.18 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_EdmaReceiveData ( flexio\_uart\_edmastate\_t \* *uartEdmaState*, uint8\_t \* *rxBuff*, uint32\_t *rxSize* )**

Parameters

|                      |                                                                    |
|----------------------|--------------------------------------------------------------------|
| <i>uartEdmaState</i> | The run-time structure of FlexIO-simulated UART.                   |
| <i>rxBuff</i>        | A pointer to the buffer containing 8-bit read data chars received. |
| <i>rxSize</i>        | The number of bytes to receive.                                    |

## Function Documentation

### Returns

An error code or `kStatus_FlexIO_UART_Success`.

### 62.3.19 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_EdmaGetReceiveStatus ( flexio\_uart\_edmastate\_t \* *uartEdmaState*, uint32\_t \* *bytesRemaining* )

#### Parameters

|                        |                                                                                                                         |
|------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>uartEdmaState</i>   | The run-time structure of FlexIO-simulated UART.                                                                        |
| <i>bytes-Remaining</i> | A pointer to a value that is populated with the number of bytes which still need to be received in the active transfer. |

### Returns

An error code or `kStatus_FlexIO_UART_Success`.

#### Return values

|                                    |                                                                                                                                |
|------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| <i>kStatus_FlexIO_UART_Success</i> | The receive has completed successfully.                                                                                        |
| <i>kStatus_FlexIO_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. |

### 62.3.20 flexio\_uart\_status\_t FLEXIO\_UART\_DRV\_EdmaAbortReceivingData ( flexio\_uart\_edmastate\_t \* *uartEdmaState* )

#### Parameters

|                      |                                                  |
|----------------------|--------------------------------------------------|
| <i>uartEdmaState</i> | The run-time structure of FlexIO-simulated UART. |
|----------------------|--------------------------------------------------|

### Returns

An error code or `kStatus_UART_Success`.

Return values

|                                                 |                                      |
|-------------------------------------------------|--------------------------------------|
| <i>kStatus_FlexIO_UART_Success</i>              | The receive was successful.          |
| <i>kStatus_FlexIO_UART_NoTransmitInProgress</i> | No receive is currently in progress. |



---

## Function Documentation

## Chapter 63

### Flexio\_spi\_hal

#### 63.1 Overview

##### Data Structures

- struct `flexio_spi_dev_t`  
*Define structure of configuring the flexio spi device. [More...](#)*
- struct `flexio_spi_master_config_t`  
*Define structure of configuring the flexio spi bus when master. [More...](#)*
- struct `flexio_spi_slave_config_t`  
*Define structure of configuring the flexio spi bus when slave. [More...](#)*

##### Functions

- `flexio_status_t FLEXIO_SPI_HAL_ConfigMaster` (`flexio_spi_dev_t *devPtr`, const `flexio_spi_master_config_t *configPtr`)  
*Configure the flexio working as spi master.*
- `flexio_status_t FLEXIO_SPI_HAL_ConfigSlave` (`flexio_spi_dev_t *devPtr`, const `flexio_spi_slave_config_t *configPtr`)  
*Configure the flexio working as spi slave.*
- `bool FLEXIO_SPI_HAL_GetTxBufferEmptyFlag` (`flexio_spi_dev_t *devPtr`)  
*Get the flag if the tx buffer is empty.*
- `void FLEXIO_SPI_HAL_ClearTxBufferEmptyFlag` (`flexio_spi_dev_t *devPtr`)  
*Clear the flag that tx buffer is empty.*
- `void FLEXIO_SPI_HAL_SetTxBufferEmptyIntCmd` (`flexio_spi_dev_t *devPtr`, `bool enable`)  
*Switch on/off the interrupt for event of tx buffer empty.*
- `bool FLEXIO_SPI_HAL_GetTxErrFlag` (`flexio_spi_dev_t *devPtr`)  
*Get the flag of tx error.*
- `void FLEXIO_SPI_HAL_ClearTxErrFlag` (`flexio_spi_dev_t *devPtr`)  
*Clear the flag of tx error manually.*
- `void FLEXIO_SPI_HAL_SetTxErrIntCmd` (`flexio_spi_dev_t *devPtr`, `bool enable`)  
*Switch on/off the interrupt for tx error event.*
- `void FLEXIO_SPI_HAL_PutDataMSB` (`flexio_spi_dev_t *devPtr`, `uint32_t dat`)  
*Put the data to tx buffer as MSB transfer.*
- `void FLEXIO_SPI_HAL_PutDataMSBPolling` (`flexio_spi_dev_t *devPtr`, `uint32_t dat`)  
*Put the data to tx buffer as MSB transfer when empty.*
- `void FLEXIO_SPI_HAL_PutDataLSB` (`flexio_spi_dev_t *devPtr`, `uint32_t dat`)  
*Put the data to tx buffer as LSB transfer.*
- `void FLEXIO_SPI_HAL_PutDataLSBPolling` (`flexio_spi_dev_t *devPtr`, `uint32_t dat`)  
*Put the data to tx buffer as LSB transfer when empty.*
- `void FLEXIO_SPI_HAL_SetTxDmaCmd` (`flexio_spi_dev_t *devPtr`, `bool enable`)  
*Switch on/off the DMA support for tx event.*
- `uint32_t FLEXIO_SPI_HAL_GetTxBufferMSBAddr` (`flexio_spi_dev_t *devPtr`)  
*Get the tx MSB buffer's register for DMA use.*
- `uint32_t FLEXIO_SPI_HAL_GetTxBufferLSBAddr` (`flexio_spi_dev_t *devPtr`)

## Data Structure Documentation

- *Get the tx LSB buffer's register for DMA use.*
- bool [FLEXIO\\_SPI\\_HAL\\_GetRxBufferFullFlag](#) ([flexio\\_spi\\_dev\\_t](#) \*devPtr)
- *Get the flag if the rx buffer is full.*
- void [FLEXIO\\_SPI\\_HAL\\_ClearRxBufferFullFlag](#) ([flexio\\_spi\\_dev\\_t](#) \*devPtr)
- *Clear the flag of rx buffer full manually.*
- void [FLEXIO\\_SPI\\_HAL\\_SetRxBufferFullIntCmd](#) ([flexio\\_spi\\_dev\\_t](#) \*devPtr, bool enable)
- *Switch on/off the interrupt of rx buffer full event.*
- bool [FLEXIO\\_SPI\\_HAL\\_GetRxErrFlag](#) ([flexio\\_spi\\_dev\\_t](#) \*devPtr)
- *Get the flag of rx error.*
- void [FLEXIO\\_SPI\\_HAL\\_ClearRxErrFlag](#) ([flexio\\_spi\\_dev\\_t](#) \*devPtr)
- *Clear the flag of rx error manually.*
- void [FLEXIO\\_SPI\\_HAL\\_SetRxErrIntCmd](#) ([flexio\\_spi\\_dev\\_t](#) \*devPtr, bool enable)
- *Switch on/off the interrupt of the rx error event.*
- uint32\_t [FLEXIO\\_SPI\\_HAL\\_GetDataMSB](#) ([flexio\\_spi\\_dev\\_t](#) \*devPtr)
- *Get the data from rx MSB buffer.*
- uint32\_t [FLEXIO\\_SPI\\_HAL\\_GetDataMSBPolling](#) ([flexio\\_spi\\_dev\\_t](#) \*devPtr)
- *Get the data from rx MSB buffer when full.*
- uint32\_t [FLEXIO\\_SPI\\_HAL\\_GetDataLSB](#) ([flexio\\_spi\\_dev\\_t](#) \*devPtr)
- *Get the data from rx LSB buffer.*
- uint32\_t [FLEXIO\\_SPI\\_HAL\\_GetDataLSBPolling](#) ([flexio\\_spi\\_dev\\_t](#) \*devPtr)
- *Get the data from rx LSB buffer when full.*
- void [FLEXIO\\_SPI\\_HAL\\_SetRxDmaCmd](#) ([flexio\\_spi\\_dev\\_t](#) \*devPtr, bool enable)
- *Switich on/off the DMA for rx event.*
- uint32\_t [FLEXIO\\_SPI\\_HAL\\_GetRxBufferMSBAddr](#) ([flexio\\_spi\\_dev\\_t](#) \*devPtr)
- *Get the address of rx MSB buffer.*
- uint32\_t [FLEXIO\\_SPI\\_HAL\\_GetRxBufferLSBAddr](#) ([flexio\\_spi\\_dev\\_t](#) \*devPtr)
- *Get the address of rx LSB buffer.*

## 63.2 Data Structure Documentation

### 63.2.1 struct flexio\_spi\_dev\_t

#### Data Fields

- FLEXIO\_Type \* [flexioBase](#)  
*FlexIO module base address.*
- uint32\_t [txPinIdx](#)  
*Output pin index.*
- uint32\_t [rxPinIdx](#)  
*Input pin index.*
- uint32\_t [sclkPinIdx](#)  
*Clock line, output for master, input for slave.*
- uint32\_t [csnPinIdx](#)  
*Chip select line, output for master, input for slave.*
- uint32\_t [shifterIdx](#) [2]  
*Shifter index.*
- uint32\_t [timerIdx](#) [2]  
*Timer index.*

### 63.2.1.0.0.69 Field Documentation

63.2.1.0.0.69.1 FLEXIO\_Type\* flexio\_spi\_dev\_t::flexioBase

63.2.1.0.0.69.2 uint32\_t flexio\_spi\_dev\_t::txPinIdx

63.2.1.0.0.69.3 uint32\_t flexio\_spi\_dev\_t::rxPinIdx

63.2.1.0.0.69.4 uint32\_t flexio\_spi\_dev\_t::sclkPinIdx

63.2.1.0.0.69.5 uint32\_t flexio\_spi\_dev\_t::csnPinIdx

63.2.1.0.0.69.6 uint32\_t flexio\_spi\_dev\_t::shifterIdx[2]

63.2.1.0.0.69.7 uint32\_t flexio\_spi\_dev\_t::timerIdx[2]

timer 0 is available for both master and slave. timer 1 would be only available for master and not used in slave mode.

## 63.2.2 struct flexio\_spi\_master\_config\_t

### Data Fields

- uint32\_t [flexioBusClk](#)  
*Clock frequency of flexio bus.*
- uint32\_t [baudrate](#)  
*Baudrate for spi bus.*
- uint32\_t [bitCount](#)  
*Bit count for each word.*
- bool [cphaOneEnable](#)  
*The phase of spi.*

### 63.2.2.0.0.70 Field Documentation

63.2.2.0.0.70.1 uint32\_t flexio\_spi\_master\_config\_t::flexioBusClk

63.2.2.0.0.70.2 uint32\_t flexio\_spi\_master\_config\_t::baudrate

63.2.2.0.0.70.3 uint32\_t flexio\_spi\_master\_config\_t::bitCount

63.2.2.0.0.70.4 bool flexio\_spi\_master\_config\_t::cphaOneEnable

## 63.2.3 struct flexio\_spi\_slave\_config\_t

### Data Fields

- uint32\_t [bitCount](#)  
*Bit count for each word.*

## Function Documentation

- bool `cphaOneEnable`  
*The phase of spi.*

### 63.2.3.0.0.71 Field Documentation

63.2.3.0.0.71.1 `uint32_t flexio_spi_slave_config_t::bitCount`

63.2.3.0.0.71.2 `bool flexio_spi_slave_config_t::cphaOneEnable`

## 63.3 Function Documentation

**63.3.1 `flexio_status_t FLEXIO_SPI_HAL_ConfigMaster ( flexio_spi_dev_t * devPtr, const flexio_spi_master_config_t * configPtr )`**

Parameters

|                  |                                         |
|------------------|-----------------------------------------|
| <i>devPtr</i>    | Pointer to the device.                  |
| <i>configPtr</i> | Pointer to the configuration structure. |

Returns

Execution status.

**63.3.2 `flexio_status_t FLEXIO_SPI_HAL_ConfigSlave ( flexio_spi_dev_t * devPtr, const flexio_spi_slave_config_t * configPtr )`**

Parameters

|                  |                                         |
|------------------|-----------------------------------------|
| <i>devPtr</i>    | Pointer to the device.                  |
| <i>configPtr</i> | Pointer to the configuration structure. |

Returns

Execution status.

**63.3.3 `bool FLEXIO_SPI_HAL_GetTxBufferEmptyFlag ( flexio_spi_dev_t * devPtr )`**



Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

Returns

Assertion of the event.

**63.3.4 void FLEXIO\_SPI\_HAL\_ClearTxBufferEmptyFlag ( flexio\_spi\_dev\_t \* *devPtr* )**

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

**63.3.5 void FLEXIO\_SPI\_HAL\_SetTxBufferEmptyIntCmd ( flexio\_spi\_dev\_t \* *devPtr*, bool *enable* )**

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
| <i>enable</i> | Switcher to the event. |

**63.3.6 bool FLEXIO\_SPI\_HAL\_GetTxErrFlag ( flexio\_spi\_dev\_t \* *devPtr* )**

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

Returns

Assertion of the event.

**63.3.7 void FLEXIO\_SPI\_HAL\_ClearTxErrFlag ( flexio\_spi\_dev\_t \* *devPtr* )**

## Function Documentation

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

### 63.3.8 void FLEXIO\_SPI\_HAL\_SetTxErrIntCmd ( flexio\_spi\_dev\_t \* *devPtr*, bool *enable* )

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
| <i>enable</i> | Switcher to the event. |

### 63.3.9 void FLEXIO\_SPI\_HAL\_PutDataMSB ( flexio\_spi\_dev\_t \* *devPtr*, uint32\_t *dat* )

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
| <i>dat</i>    | Sending data.          |

### 63.3.10 void FLEXIO\_SPI\_HAL\_PutDataMSBPolling ( flexio\_spi\_dev\_t \* *devPtr*, uint32\_t *dat* )

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
| <i>dat</i>    | Sending data.          |

### 63.3.11 void FLEXIO\_SPI\_HAL\_PutDataLSB ( flexio\_spi\_dev\_t \* *devPtr*, uint32\_t *dat* )

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
| <i>dat</i>    | Sending data.          |

**63.3.12 void FLEXIO\_SPI\_HAL\_PutDataLSBPolling ( flexio\_spi\_dev\_t \* *devPtr*, uint32\_t *dat* )**

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
| <i>dat</i>    | Sending data.          |

**63.3.13 void FLEXIO\_SPI\_HAL\_SetTxDmaCmd ( flexio\_spi\_dev\_t \* *devPtr*, bool *enable* )**

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
| <i>enable</i> | Switcher to the event. |

**63.3.14 uint32\_t FLEXIO\_SPI\_HAL\_GetTxBufferMSBAddr ( flexio\_spi\_dev\_t \* *devPtr* )**

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

Returns

Address of tx MSB buffer.

**63.3.15 uint32\_t FLEXIO\_SPI\_HAL\_GetTxBufferLSBAddr ( flexio\_spi\_dev\_t \* *devPtr* )**

## Function Documentation

### Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

### Returns

Address of tx LSB buffer.

### 63.3.16 **bool FLEXIO\_SPI\_HAL\_GetRxBufferFullFlag ( flexio\_spi\_dev\_t \* devPtr )**

### Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

### Returns

Assertion of event.

### 63.3.17 **void FLEXIO\_SPI\_HAL\_ClearRxBufferFullFlag ( flexio\_spi\_dev\_t \* devPtr )**

### Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

### 63.3.18 **void FLEXIO\_SPI\_HAL\_SetRxBufferFullIntCmd ( flexio\_spi\_dev\_t \* devPtr, bool enable )**

### Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
| <i>enable</i> | Switcher to the event. |

### 63.3.19 **bool FLEXIO\_SPI\_HAL\_GetRxErrFlag ( flexio\_spi\_dev\_t \* devPtr )**

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

Returns

Assertion of event.

**63.3.20 void FLEXIO\_SPI\_HAL\_ClearRxErrFlag ( flexio\_spi\_dev\_t \* *devPtr* )**

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

**63.3.21 void FLEXIO\_SPI\_HAL\_SetRxErrIntCmd ( flexio\_spi\_dev\_t \* *devPtr*, bool *enable* )**

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
| <i>enable</i> | Switcher to the event. |

**63.3.22 uint32\_t FLEXIO\_SPI\_HAL\_GetDataMSB ( flexio\_spi\_dev\_t \* *devPtr* )**

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

Returns

Data from rx MSB buffer.

**63.3.23 uint32\_t FLEXIO\_SPI\_HAL\_GetDataMSBPolling ( flexio\_spi\_dev\_t \* *devPtr* )**

## Function Documentation

### Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

### Returns

Data from rx MSB buffer.

### 63.3.24 `uint32_t FLEXIO_SPI_HAL_GetDataLSB ( flexio_spi_dev_t * devPtr )`

### Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

### Returns

Data from rx LSB buffer.

### 63.3.25 `uint32_t FLEXIO_SPI_HAL_GetDataLSBPolling ( flexio_spi_dev_t * devPtr )`

### Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

### Returns

Data from rx LSB buffer.

### 63.3.26 `void FLEXIO_SPI_HAL_SetRxDmaCmd ( flexio_spi_dev_t * devPtr, bool enable )`

### Parameters

---

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
| <i>enable</i> | Switcher to the event. |

**63.3.27** `uint32_t FLEXIO_SPI_HAL_GetRxBufferMSBAddr ( flexio_spi_dev_t * devPtr )`

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

Returns

Address of rx MSB buffer.

**63.3.28** `uint32_t FLEXIO_SPI_HAL_GetRxBufferLSBAddr ( flexio_spi_dev_t * devPtr )`

Parameters

|               |                        |
|---------------|------------------------|
| <i>devPtr</i> | Pointer to the device. |
|---------------|------------------------|

Returns

Address of rx MSB buffer.



---

## Function Documentation



## Chapter 64

### Pwm\_hal

#### 64.1 Overview

##### Data Structures

- struct [pwm\\_module\\_setup\\_t](#)  
*Structure is used to hold the parameters to configure a PWM module. [More...](#)*
- struct [pwm\\_fault\\_setup\\_t](#)  
*Structure is used to hold the parameters to configure a PWM fault. [More...](#)*
- struct [pwm\\_capture\\_setup\\_t](#)  
*Structure is used to hold parameters to configure the capture capability of a signal pin. [More...](#)*

##### Enumerations

- enum [pwm\\_module\\_t](#) {  
    [kFlexPwmModule0](#) = 0U,  
    [kFlexPwmModule1](#),  
    [kFlexPwmModule2](#),  
    [kFlexPwmModule3](#) }  
*PWM submodules.*
- enum [pwm\\_module\\_signal\\_t](#)  
*PWM signals from each module.*
- enum [pwm\\_val\\_regs\\_t](#) {  
    [kFlexPwmVAL0](#) = 0U,  
    [kFlexPwmVAL1](#),  
    [kFlexPwmVAL2](#),  
    [kFlexPwmVAL3](#),  
    [kFlexPwmVAL4](#),  
    [kFlexPwmVAL5](#) }  
*PWM value registers.*
- enum [pwm\\_status\\_t](#) {  
    [kStatusPwmSuccess](#) = 0U,  
    [kStatusPwmError](#) = 1U,  
    [kStatusPwmInvalidArgument](#) = 2U }  
*PWM status.*
- enum [pwm\\_clock\\_src\\_t](#) {  
    [kClkSrcPwmIPBusClk](#) = 0U,  
    [kClkSrcPwmExtClk](#),  
    [kClkSrcPwm0Clk](#) }  
*PWM clock source selection.*
- enum [pwm\\_clock\\_ps\\_t](#) {

## Overview

```
kPwmDividedBy1 = 0U,
kPwmDividedBy2,
kPwmDividedBy4,
kPwmDividedBy8,
kPwmDividedBy16,
kPwmDividedBy32,
kPwmDividedBy64,
kPwmDividedBy128 }
```

*PWM prescaler factor selection for clock source.*

- enum `pwm_force_output_trigger_t` {
 

```
kForceOutputLocalForce = 0U,
kForceOutputMasterForce,
kForceOutputLocalReload,
kForceOutputMasterReload,
kForceOutputLocalSync,
kForceOutputMasterSync,
kForceOutputExternalForce }
```

*Options that can trigger a PWM FORCE\_OUT.*

- enum `pwm_init_src_t` {
 

```
kInitSrcLocalSync = 0U,
kInitSrcMasterReload,
kInitSrcMasterSync,
kInitSrcExtSync }
```

*PWM counter initialization options.*

- enum `pwm_load_frequency_t` {
 

```
kPwmLoadEvery1Opportunity = 0U,
kPwmLoadEvery2Opportunity,
kPwmLoadEvery3Opportunity,
kPwmLoadEvery4Opportunity,
kPwmLoadEvery5Opportunity,
kPwmLoadEvery6Opportunity,
kPwmLoadEvery7Opportunity,
kPwmLoadEvery8Opportunity,
kPwmLoadEvery9Opportunity,
kPwmLoadEvery10Opportunity,
kPwmLoadEvery11Opportunity,
kPwmLoadEvery12Opportunity,
kPwmLoadEvery13Opportunity,
kPwmLoadEvery14Opportunity,
kPwmLoadEvery15Opportunity,
kPwmLoadEvery16Opportunity }
```

*PWM load frequency selection.*

- enum `pwm_fault_input_t` {
 

```
kFlexPwmFault0 = 0U,
kFlexPwmFault1,
kFlexPwmFault2,
```

- kFlexPwmFault3 }
    - PWM fault select.*
- enum `pwm_capture_edge_t` {
  - kCaptureDisable = 0U,
  - kCaptureFallingEdges,
  - kCaptureRisingEdges,
  - kCaptureAnyEdges }
  - PWM capture edge select.*
- enum `pwm_force_signal_t` {
  - kFlexPwmUsePwm = 0U,
  - kFlexPwmInvertedPwm,
  - kFlexPwmSoftwareControl,
  - kFlexPwmUseExternal }
  - PWM output options when a FORCE\_OUT signal is asserted.*
- enum `pwm_chnl_pair_operation_t` {
  - kFlexPwmIndependent = 0U,
  - kFlexPwmComplementaryPwmA,
  - kFlexPwmComplementaryPwmB }
  - Options available for the PWM A & B pair operation.*
- enum `pwm_reg_reload_t` {
  - kFlexPwmReloadImmediate = 0U,
  - kFlexPwmReloadPwmHalfCycle,
  - kFlexPwmReloadPwmFullCycle,
  - kFlexPwmReloadPwmHalfAndFullCycle }
  - Options available on how to load the buffered-registers with new values.*
- enum `pwm_fault_recovery_mode_t` {
  - kFlexPwmNoRecovery = 0U,
  - kFlexPwmRecoverHalfCycle,
  - kFlexPwmRecoverFullCycle,
  - kFlexPwmRecoverHalfAndFullCycle }
  - Options available on how to re-enable the PWM output when recovering from a fault.*

## Functions

- void `PWM_HAL_Init` (PWM\_Type \*base)
  - Initialize the PWM to its reset state.*
- void `PWM_HAL_SetupPwmSubModule` (PWM\_Type \*base, `pwm_module_t` subModuleNum, `pwm_module_setup_t` \*setupParams)
  - Sets up a PWM sub-module.*
- void `PWM_HAL_SetupFaults` (PWM\_Type \*base, `pwm_fault_input_t` faultNum, `pwm_fault_setup_t` \*setupParams)
  - Sets up the working of the Flex PWM fault protection.*
- void `PWM_HAL_SetupCapture` (PWM\_Type \*base, `pwm_module_t` subModuleNum, `pwm_module_signal_t` pwmSignal, `pwm_capture_setup_t` \*setupParams)
  - Sets up the Flex PWM capture.*
- uint16\_t `PWM_HAL_GetCaptureValReg` (PWM\_Type \*base, `pwm_module_t` subModuleNum, `pwm_val_regs_t` cmpReg)
  - Gets PWM capture value.*

## Overview

- void [PWM\\_HAL\\_SetValReg](#) (PWM\_Type \*base, uint8\_t subModuleNum, [pwm\\_val\\_regs\\_t](#) valReg, uint16\_t val)  
*Sets PWM value register.*
- void [PWM\\_HAL\\_SetupForceSignal](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum, [pwm\\_module\\_signal\\_t](#) pwmSignal, [pwm\\_force\\_signal\\_t](#) mode)  
*Selects the signal to output when a FORCE\_OUT signal is asserted.*
- static uint16\_t [PWM\\_HAL\\_GetCounter](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum)  
*Returns PWM peripheral current counter value.*
- static void [PWM\\_HAL\\_SetCounterInitVal](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum, uint16\_t val)  
*Sets PWM timer counter initial value.*
- static void [PWM\\_HAL\\_SetForceCmd](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum, bool val)  
*Outputs a FORCE signal.*
- static void [PWM\\_HAL\\_SetOutputPolarityPwmBCmd](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum, bool val)  
*Sets output polarity for PWM\_B.*
- static void [PWM\\_HAL\\_SetOutputPolarityPwmACmd](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum, bool val)  
*Sets output polarity for PWM\_A.*
- static void [PWM\\_HAL\\_SetOutputPolarityPwmXCmd](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum, bool val)  
*Sets output polarity for PWM\_X.*
- static void [PWM\\_HAL\\_SetOutputTriggerCmd](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum, uint8\_t valueReg, bool val)  
*Enables or disables if a match with a value register will cause an output trigger.*
- static void [PWM\\_HAL\\_SetPwmAFaultInputCmd](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum, [pwm\\_fault\\_input\\_t](#) fault, bool val)  
*Enables or disables fault input for PWM A.*
- static void [PWM\\_HAL\\_SetPwmBFaultInputCmd](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum, [pwm\\_fault\\_input\\_t](#) fault, bool val)  
*Enables or disables fault input for PWM B.*
- static void [PWM\\_HAL\\_SetPwmXFaultInputCmd](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum, [pwm\\_fault\\_input\\_t](#) fault, bool val)  
*Enables or disables fault input for PWM X.*
- static void [PWM\\_HAL\\_SetOutputPwmXCmd](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum, bool val)  
*Sets PWM\_X pin to input or output.*
- static void [PWM\\_HAL\\_SetOutputPwmBCmd](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum, bool val)  
*Sets PWM\_B pin to input or output.*
- static void [PWM\\_HAL\\_SetOutputPwmACmd](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum, bool val)  
*Sets PWM\_A pin to input or output.*
- static void [PWM\\_HAL\\_SetSwCtrlOutCmd](#) (PWM\_Type \*base, [pwm\\_module\\_t](#) subModuleNum, [pwm\\_module\\_signal\\_t](#) output, bool val)  
*Sets software control output for a pin to high or low.*
- static void [PWM\\_HAL\\_SetPwmRunCmd](#) (PWM\_Type \*base, uint8\_t subModules, bool val)  
*Sets PWM generator run.*

- static void [PWM\\_HAL\\_SetFaultIntCmd](#) (PWM\_Type \*base, [pwm\\_fault\\_input\\_t](#) fault, bool val)  
*Sets fault interrupt.*
- static void [PWM\\_HAL\\_ClearFaultFlags](#) (PWM\_Type \*base, [pwm\\_fault\\_input\\_t](#) fault)  
*Clears fault flags.*

## 64.2 Data Structure Documentation

### 64.2.1 struct pwm\_module\_setup\_t

#### Data Fields

- [pwm\\_init\\_src\\_t](#) [cntrInitSel](#)  
*Option to initialize the counter.*
- [pwm\\_clock\\_src\\_t](#) [clkSrc](#)  
*Clock source for the counter.*
- [pwm\\_clock\\_ps\\_t](#) [prescale](#)  
*Pre-scaler to divide down the clock.*
- [pwm\\_chnl\\_pair\\_operation\\_t](#) [chnlPairOper](#)  
*Channel pair in independent or complementary mode.*
- [pwm\\_reg\\_reload\\_t](#) [reloadLogic](#)  
*PWM Reload logic setup.*
- [pwm\\_load\\_frequency\\_t](#) [reloadFreq](#)  
*Specifies when to reload, used when user's choice is not immediate reload.*
- [pwm\\_force\\_output\\_trigger\\_t](#) [forceTrig](#)  
*Specify which signal will trigger a FORCE\_OUT.*

### 64.2.2 struct pwm\_fault\_setup\_t

#### Data Fields

- bool [automaticClearing](#)  
*true: Use automatic fault clearing; false: Manual fault clearing.*
- bool [faultLevel](#)  
*true: Logic 1 indicates fault; false: Logic 0 indicates fault.*
- bool [useFaultFilter](#)  
*true: Use the filtered fault signal; false: Use the direct path from fault input.*
- [pwm\\_fault\\_recovery\\_mode\\_t](#) [recMode](#)  
*Specify when to re-enable the PWM output.*

### 64.2.3 struct pwm\_capture\_setup\_t

#### Data Fields

- bool [captureInputSel](#)  
*true: Use the edge counter signal as source false: Use the raw input signal from the pin as source*
- [uint8\\_t](#) [edgeCompareVal](#)

## Enumeration Type Documentation

- *Compare value, used only if edge counter is used as source.*  
[pwm\\_capture\\_edge\\_t edge0](#)  
*Specify which edge causes a capture for input circuitry 0.*
- [pwm\\_capture\\_edge\\_t edge1](#)  
*Specify which edge causes a capture for input circuitry 1.*
- bool [oneShotCapture](#)  
*true: Use one-shot capture mode; false: Use free-running capture mode*

## 64.3 Enumeration Type Documentation

### 64.3.1 enum pwm\_module\_t

Enumerator

- kFlexPwmModule0* Sub-module 0.
- kFlexPwmModule1* Sub-module 1.
- kFlexPwmModule2* Sub-module 2.
- kFlexPwmModule3* Sub-module 3.

### 64.3.2 enum pwm\_val\_regs\_t

Enumerator

- kFlexPwmVAL0* PWM VAL0 reg.
- kFlexPwmVAL1* PWM VAL1 reg.
- kFlexPwmVAL2* PWM VAL2 reg.
- kFlexPwmVAL3* PWM VAL3 reg.
- kFlexPwmVAL4* PWM VAL4 reg.
- kFlexPwmVAL5* PWM VAL5 reg.

### 64.3.3 enum pwm\_status\_t

Enumerator

- kStatusPwmSuccess* PWM success status.
- kStatusPwmError* PWM error status.
- kStatusPwmInvalidArgument* PWM invalid argument.

### 64.3.4 enum pwm\_clock\_src\_t

Enumerator

- kClkSrcPwmIPBusClk* The IPBus clock is used as the clock.

***kClkSrcPwmExtClk*** EXT\_CLK is used as the clock.

***kClkSrcPwm0Clk*** Clock of Submodule 0 (AUX\_CLK) is used as the source clock.

### 64.3.5 enum pwm\_clock\_ps\_t

Enumerator

***kPwmDividedBy1*** PWM clock frequency = fclk/1.

***kPwmDividedBy2*** PWM clock frequency = fclk/2.

***kPwmDividedBy4*** PWM clock frequency = fclk/4.

***kPwmDividedBy8*** PWM clock frequency = fclk/8.

***kPwmDividedBy16*** PWM clock frequency = fclk/16.

***kPwmDividedBy32*** PWM clock frequency = fclk/32.

***kPwmDividedBy64*** PWM clock frequency = fclk/64.

***kPwmDividedBy128*** PWM clock frequency = fclk/128.

### 64.3.6 enum pwm\_force\_output\_trigger\_t

Enumerator

***kForceOutputLocalForce*** The local force signal, CTRL2[FORCE], from this submodule is used to force updates.

***kForceOutputMasterForce*** The master force signal from submodule 0 is used to force updates.

***kForceOutputLocalReload*** The local reload signal from this submodule is used to force updates without regard to the state of LDOK.

***kForceOutputMasterReload*** The master reload signal from submodule 0 is used to force updates if LDOK is set.

***kForceOutputLocalSync*** The local sync signal from this submodule is used to force updates.

***kForceOutputMasterSync*** The master sync signal from submodule 0 is used to force updates.

***kForceOutputExternalForce*** The external force signal, EXT\_FORCE, from outside the PWM module causes updates.

### 64.3.7 enum pwm\_init\_src\_t

Enumerator

***kInitSrcLocalSync*** Local sync (PWM\_X) causes initialization.

***kInitSrcMasterReload*** Master reload from submodule 0 causes initialization.

***kInitSrcMasterSync*** Master sync from submodule 0 causes initialization.

***kInitSrcExtSync*** EXT\_SYNC causes initialization.

## Enumeration Type Documentation

### 64.3.8 enum pwm\_load\_frequency\_t

Enumerator

|                                   |                             |
|-----------------------------------|-----------------------------|
| <i>kPwmLoadEvery1Opportunity</i>  | Every 1 PWM opportunity.    |
| <i>kPwmLoadEvery2Opportunity</i>  | Every 2 PWM opportunities.  |
| <i>kPwmLoadEvery3Opportunity</i>  | Every 3 PWM opportunities.  |
| <i>kPwmLoadEvery4Opportunity</i>  | Every 4 PWM opportunities.  |
| <i>kPwmLoadEvery5Opportunity</i>  | Every 5 PWM opportunities.  |
| <i>kPwmLoadEvery6Opportunity</i>  | Every 6 PWM opportunities.  |
| <i>kPwmLoadEvery7Opportunity</i>  | Every 7 PWM opportunities.  |
| <i>kPwmLoadEvery8Opportunity</i>  | Every 8 PWM opportunities.  |
| <i>kPwmLoadEvery9Opportunity</i>  | Every 9 PWM opportunities.  |
| <i>kPwmLoadEvery10Opportunity</i> | Every 10 PWM opportunities. |
| <i>kPwmLoadEvery11Opportunity</i> | Every 11 PWM opportunities. |
| <i>kPwmLoadEvery12Opportunity</i> | Every 12 PWM opportunities. |
| <i>kPwmLoadEvery13Opportunity</i> | Every 13 PWM opportunities. |
| <i>kPwmLoadEvery14Opportunity</i> | Every 14 PWM opportunities. |
| <i>kPwmLoadEvery15Opportunity</i> | Every 15 PWM opportunities. |
| <i>kPwmLoadEvery16Opportunity</i> | Every 16 PWM opportunities. |

### 64.3.9 enum pwm\_fault\_input\_t

Enumerator

|                       |                    |
|-----------------------|--------------------|
| <i>kFlexPwmFault0</i> | Fault 0 input pin. |
| <i>kFlexPwmFault1</i> | Fault 1 input pin. |
| <i>kFlexPwmFault2</i> | Fault 2 input pin. |
| <i>kFlexPwmFault3</i> | Fault 3 input pin. |

### 64.3.10 enum pwm\_capture\_edge\_t

Enumerator

|                             |                        |
|-----------------------------|------------------------|
| <i>kCaptureDisable</i>      | Disabled.              |
| <i>kCaptureFallingEdges</i> | Capture falling edges. |
| <i>kCaptureRisingEdges</i>  | Capture rising edges.  |
| <i>kCaptureAnyEdges</i>     | Capture any edge.      |



### 64.3.11 enum pwm\_force\_signal\_t

Enumerator

- kFlexPwmUsePwm* Generated PWM signal is used by the deadtime logic.
- kFlexPwmInvertedPwm* Inverted PWM signal is used by the deadtime logic.
- kFlexPwmSoftwareControl* Software controlled value is used by the deadtime logic.
- kFlexPwmUseExternal* PWM\_EXT\_A signal is used by the deadtime logic.

### 64.3.12 enum pwm\_chnl\_pair\_operation\_t

Enumerator

- kFlexPwmIndependent* PWM A & PWM B operation as 2 independent channels.
- kFlexPwmComplementaryPwmA* PWM A & PWM B are complementary channels, PWM A generates the signal.
- kFlexPwmComplementaryPwmB* PWM A & PWM B are complementary channels, PWM B generates the signal.

### 64.3.13 enum pwm\_reg\_reload\_t

Enumerator

- kFlexPwmReloadImmediate* Buffered-registers get loaded with new values as soon as LDOK bit is set.
- kFlexPwmReloadPwmHalfCycle* Registers loaded on a PWM half cycle.
- kFlexPwmReloadPwmFullCycle* Registers loaded on a PWM full cycle.
- kFlexPwmReloadPwmHalfAndFullCycle* Registers loaded on a PWM half & full cycle.

### 64.3.14 enum pwm\_fault\_recovery\_mode\_t

Enumerator

- kFlexPwmNoRecovery* PWM output will stay inactive.
- kFlexPwmRecoverHalfCycle* PWM output re-enabled at the first half cycle.
- kFlexPwmRecoverFullCycle* PWM output re-enabled at the first full cycle.
- kFlexPwmRecoverHalfAndFullCycle* PWM output re-enabled at the first half or full cycle.

## 64.4 Function Documentation

### 64.4.1 void PWM\_HAL\_Init ( PWM\_Type \* *base* )

Set the control registers to their reset state

## Function Documentation

Parameters

|             |                                         |
|-------------|-----------------------------------------|
| <i>base</i> | Base address pointer of eflexPWM module |
|-------------|-----------------------------------------|

### 64.4.2 void PWM\_HAL\_SetupPwmSubModule ( PWM\_Type \* *base*, pwm\_module\_t *subModuleNum*, pwm\_module\_setup\_t \* *setupParams* )

Flex PWM has 4 sub-modules. This function sets up key features that configure the working of each sub-module. This function will setup:

1. Clock source and clock prescaler
2. Submodules PWM A & PWM B signals operation (independent or complementary)
3. Reload logic to use and reload frequency
4. Force trigger to use to generate the FORCE\_OUT signal.

Parameters

|                      |                                             |
|----------------------|---------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module     |
| <i>subModule-Num</i> | is a number of the PWM submodule.           |
| <i>setupParams</i>   | Parameters passed in to setup the submodule |

### 64.4.3 void PWM\_HAL\_SetupFaults ( PWM\_Type \* *base*, pwm\_fault\_input\_t *faultNum*, pwm\_fault\_setup\_t \* *setupParams* )

Flex PWM has 4 fault inputs. This function sets up the working of each fault. The function will setup:

1. Fault automatic clearing function
2. Sets up the fault level
3. Defines if the fault filter should be used for this fault input
4. Recovery mode to be used to re-enable the PWM output

Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>base</i>     | Base address pointer of eflexPWM module    |
| <i>faultNum</i> | is a number of the PWM fault to configure. |

|                    |                                         |
|--------------------|-----------------------------------------|
| <i>setupParams</i> | Parameters passed in to setup the fault |
|--------------------|-----------------------------------------|

**64.4.4 void PWM\_HAL\_SetupCapture ( PWM\_Type \* *base*, pwm\_module\_t *subModuleNum*, pwm\_module\_signal\_t *pwmSignal*, pwm\_capture\_setup\_t \* *setupParams* )**

Each PWM submodule has 3 pins can be configured to use for capture. This function will setup the capture for each pin as follows:

1. Whether to use the edge counter or raw input
2. Edge capture mode
3. One-shot or continuous

Parameters

|                      |                                             |
|----------------------|---------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module     |
| <i>subModule-Num</i> | is a number of the PWM submodule.           |
| <i>pwmSignal</i>     | Which signal in the submodule to setup      |
| <i>setupParams</i>   | Parameters passed in to setup the input pin |

**64.4.5 uint16\_t PWM\_HAL\_GetCaptureValReg ( PWM\_Type \* *base*, pwm\_module\_t *subModuleNum*, pwm\_val\_regs\_t *cmpReg* )**

Read one of the 6 capture value registers

Parameters

|                      |                                              |
|----------------------|----------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module      |
| <i>subModule-Num</i> | is a number of the PWM submodule.            |
| <i>cmpReg</i>        | is a number of value compare register to get |

## Function Documentation

Returns

PWM value register

**64.4.6 void PWM\_HAL\_SetValReg ( PWM\_Type \* *base*, uint8\_t *subModuleNum*,  
pwm\_val\_regs\_t *valReg*, uint16\_t *val* )**

Sets one of the 6 value registers.

Parameters

|                      |                                               |
|----------------------|-----------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module       |
| <i>subModule-Num</i> | is a number of the PWM submodule.             |
| <i>valReg</i>        | is the number of the value register to be set |
| <i>val</i>           | is a number of value to write                 |

**64.4.7 void PWM\_HAL\_SetupForceSignal ( PWM\_Type \* *base*, pwm\_module\_t *subModuleNum*, pwm\_module\_signal\_t *pwmSignal*, pwm\_force\_signal\_t *mode* )**

User specifies which pin to configure by supplying the submodule number and whether he wishes to modify PWM A or PWM B within that submodule

Parameters

|                      |                                                   |
|----------------------|---------------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module           |
| <i>subModule-Num</i> | is a number of the PWM submodule.                 |
| <i>pwmSignal</i>     | specifies which signal to work with in the module |
| <i>mode</i>          | signal to output when a FORCE_OUT is triggered    |

**64.4.8 static uint16\_t PWM\_HAL\_GetCounter ( PWM\_Type \* *base*, pwm\_module\_t *subModuleNum* ) [inline], [static]**

Parameters

|                      |                                         |
|----------------------|-----------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module |
| <i>subModule-Num</i> | is a number of the PWM submodule.       |

## Function Documentation

Returns

current PWM counter value

**64.4.9** `static void PWM_HAL_SetCounterInitVal ( PWM_Type * base,  
pwm_module_t subModuleNum, uint16_t val ) [inline], [static]`

Parameters

|                      |                                         |
|----------------------|-----------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module |
| <i>subModule-Num</i> | is a number of the PWM submodule.       |
| <i>val</i>           | initial value to be set                 |

**64.4.10 static void PWM\_HAL\_SetForceCmd ( PWM\_Type \* *base*, pwm\_module\_t *subModuleNum*, bool *val* ) [inline], [static]**

This function will enable/disable the force init logic and assert/de-assert the FORCE signal

Parameters

|                      |                                         |
|----------------------|-----------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module |
| <i>subModule-Num</i> | is a number of the PWM submodule.       |
| <i>val</i>           | true to enable, false to disable.       |

**64.4.11 static void PWM\_HAL\_SetOutputPolarityPwmBCmd ( PWM\_Type \* *base*, pwm\_module\_t *subModuleNum*, bool *val* ) [inline], [static]**

Parameters

|                      |                                                                |
|----------------------|----------------------------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module                        |
| <i>subModule-Num</i> | is a number of the PWM submodule.                              |
| <i>val</i>           | true to set inverted output, false to set non inverted output. |

**64.4.12 static void PWM\_HAL\_SetOutputPolarityPwmACmd ( PWM\_Type \* *base*, pwm\_module\_t *subModuleNum*, bool *val* ) [inline], [static]**

## Function Documentation

### Parameters

|                      |                                                                |
|----------------------|----------------------------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module                        |
| <i>subModule-Num</i> | is a number of the PWM submodule.                              |
| <i>val</i>           | true to set inverted output, false to set non inverted output. |

**64.4.13** `static void PWM_HAL_SetOutputPolarityPwmXCmd ( PWM_Type * base, pwm_module_t subModuleNum, bool val ) [inline], [static]`

### Parameters

|                      |                                                                |
|----------------------|----------------------------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module                        |
| <i>subModule-Num</i> | is a number of the PWM submodule.                              |
| <i>val</i>           | true to set inverted output, false to set non inverted output. |

**64.4.14** `static void PWM_HAL_SetOutputTriggerCmd ( PWM_Type * base, pwm_module_t subModuleNum, uint8_t valueReg, bool val ) [inline], [static]`

There are 2 triggers available per PWM submodule. This function allows the user the ability to activate a trigger when the counter matches one of the 6 value registers. Enabling VAL0, VAL2 or VAL4 will output a trigger on a match on TRIG0. Enabling VAL1, VAL3, VAL5 will output a trigger on a match on TRIG1.

### Parameters

|                      |                                                    |
|----------------------|----------------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module            |
| <i>subModule-Num</i> | is a number of the PWM submodule.                  |
| <i>valueReg</i>      | register that is the cause for the output trigger. |
| <i>val</i>           | true to trigger enable, false to disable.          |



**64.4.15** `static void PWM_HAL_SetPwmAFaultInputCmd ( PWM_Type * base,  
pwm_module_t subModuleNum, pwm_fault_input_t fault, bool val )`  
`[inline], [static]`

Enabling the specified fault will cause the PWM A signal to deactivate when the fault occurs. User should configure the PWM faults by calling [PWM\\_HAL\\_SetupFaults\(\)](#) prior to enabling them in the submodules.

## Function Documentation

### Parameters

|                      |                                                               |
|----------------------|---------------------------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module                       |
| <i>subModule-Num</i> | is a number of the PWM submodule.                             |
| <i>fault</i>         | number, options: 0,1,2,3 .                                    |
| <i>val</i>           | true to enable the fault input, false to disable fault input. |

**64.4.16** `static void PWM_HAL_SetPwmBFaultInputCmd ( PWM_Type * base, pwm_module_t subModuleNum, pwm_fault_input_t fault, bool val )`  
**[inline], [static]**

Enabling the specified fault will cause the PWM B signal to deactivate when the fault occurs. User should configure the PWM faults by calling [PWM\\_HAL\\_SetupFaults\(\)](#) prior to enabling them in the submodules.

### Parameters

|                      |                                                               |
|----------------------|---------------------------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module                       |
| <i>subModule-Num</i> | is a number of the PWM submodule.                             |
| <i>fault</i>         | number, options: 0,1,2,3 .                                    |
| <i>val</i>           | true to enable the fault input, false to disable fault input. |

**64.4.17** `static void PWM_HAL_SetPwmXFaultInputCmd ( PWM_Type * base, pwm_module_t subModuleNum, pwm_fault_input_t fault, bool val )`  
**[inline], [static]**

Enabling the specified fault will cause the PWM X signal to deactivate when the fault occurs. User should configure the PWM faults by calling [PWM\\_HAL\\_SetupFaults\(\)](#) prior to enabling them in the submodules.

### Parameters

|                      |                                         |
|----------------------|-----------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module |
| <i>subModule-Num</i> | is a number of the PWM submodule.       |

|              |                                                                  |
|--------------|------------------------------------------------------------------|
| <i>fault</i> | number, options: 0,1,2,3.                                        |
| <i>val</i>   | true to enable the fault input; false to disable the fault input |

**64.4.18** `static void PWM_HAL_SetOutputPwmXCmd ( PWM_Type * base,  
pwm_module_t subModuleNum, bool val ) [inline], [static]`

Parameters

|                      |                                                                       |
|----------------------|-----------------------------------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module                               |
| <i>subModule-Num</i> | Number of the PWM submodule.                                          |
| <i>val</i>           | true to make the pin as output output, false to make the pin as input |

**64.4.19** `static void PWM_HAL_SetOutputPwmBCmd ( PWM_Type * base,  
pwm_module_t subModuleNum, bool val ) [inline], [static]`

Parameters

|                      |                                                                       |
|----------------------|-----------------------------------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module                               |
| <i>subModule-Num</i> | Number of the PWM submodule.                                          |
| <i>val</i>           | true to make the pin as output output, false to make the pin as input |

**64.4.20** `static void PWM_HAL_SetOutputPwmACmd ( PWM_Type * base,  
pwm_module_t subModuleNum, bool val ) [inline], [static]`

Parameters

|                      |                                                                       |
|----------------------|-----------------------------------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module                               |
| <i>subModule-Num</i> | Number of the PWM submodule.                                          |
| <i>val</i>           | true to make the pin as output output, false to make the pin as input |

---

## Function Documentation

**64.4.21** `static void PWM_HAL_SetSwCtrlOutCmd ( PWM_Type * base,  
pwm_module_t subModuleNum, pwm_module_signal_t output, bool val )`  
`[inline], [static]`

User specifies which signal to modify by supplying the submodule number and whether he wishes to modify PWM A or PWM B within that submodule

Parameters

|                      |                                                                           |
|----------------------|---------------------------------------------------------------------------|
| <i>base</i>          | Base address pointer of eflexPWM module                                   |
| <i>subModule-Num</i> | is a number of the PWM submodule.                                         |
| <i>output</i>        | specifies which signal to work with in the module, 0 is PWM B, 1 is PWM A |
| <i>val</i>           | true to supply a logic 1, false to supply a logic 0.                      |

**64.4.22 static void PWM\_HAL\_SetPwmRunCmd ( PWM\_Type \* *base*, uint8\_t *subModules*, bool *val* ) [inline], [static]**

Parameters

|                   |                                                                                  |
|-------------------|----------------------------------------------------------------------------------|
| <i>base</i>       | Base address pointer of eflexPWM module                                          |
| <i>subModules</i> | represented by corresponded bits.                                                |
| <i>val</i>        | true to run selected subModuleNums, false to stop selected subModuleNums output. |

**64.4.23 static void PWM\_HAL\_SetFaultIntCmd ( PWM\_Type \* *base*, pwm\_fault\_input\_t *fault*, bool *val* ) [inline], [static]**

Parameters

|              |                                                         |
|--------------|---------------------------------------------------------|
| <i>base</i>  | Base address pointer of eflexPWM module                 |
| <i>fault</i> | represented by corresponded bits.                       |
| <i>val</i>   | true to enable the interrupt request, false to disable. |

**64.4.24 static void PWM\_HAL\_ClearFaultFlags ( PWM\_Type \* *base*, pwm\_fault\_input\_t *fault* ) [inline], [static]**

Parameters

---

## Function Documentation

|              |                                         |
|--------------|-----------------------------------------|
| <i>base</i>  | Base address pointer of eflexPWM module |
| <i>fault</i> | represented by corresponded bits.       |

## 65 Revision History

This table summarizes revisions to this document.

| Revision History |         |                           |
|------------------|---------|---------------------------|
| Revision number  | Date    | Substantial changes       |
| 0                | 04/2015 | Kinetis SDK 1.2.0 release |



**How to Reach Us:**

**Home Page:**

[freescale.com](http://freescale.com)

**Web Support:**

[freescale.com/support](http://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](http://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.

© 2015 Freescale Semiconductor, Inc.