

# S32SDK User Manual

## S32K1xx RTM 4.0.2

Generated by Doxygen 1.8.10

Fri Jun 11 2021 08:16:20

## Contents

<b>1</b>	<b>S32 SDK</b>	<b>1</b>
<b>2</b>	<b>Components</b>	<b>2</b>
<b>3</b>	<b>PAL vs PD usage</b>	<b>4</b>
<b>4</b>	<b>Supported Platforms</b>	<b>5</b>
<b>5</b>	<b>Installation</b>	<b>5</b>
<b>6</b>	<b>Build Tools</b>	<b>5</b>
<b>7</b>	<b>IDE Support</b>	<b>6</b>
<b>8</b>	<b>Configuration</b>	<b>7</b>
<b>9</b>	<b>Acronyms and Abbreviations</b>	<b>7</b>
<b>10</b>	<b>MISRA Compliance</b>	<b>7</b>
<b>11</b>	<b>Development guidelines</b>	<b>8</b>
<b>12</b>	<b>Error detection and reporting</b>	<b>8</b>
<b>13</b>	<b>Examples and Demos</b>	<b>9</b>
13.1	Introduction . . . . .	9
13.2	Usage . . . . .	9
13.2.1	How to build . . . . .	9
13.2.2	How to debug . . . . .	10
13.2.3	Using terminal emulator . . . . .	11
13.3	Demo Applications . . . . .	12
13.3.1	Hello World . . . . .	13
13.3.2	Hello World - IAR Embedded Workbench . . . . .	14
13.3.3	Hello World - Makefile . . . . .	15
13.3.4	FlexCAN Encrypted . . . . .	16
13.3.5	CSEC BOOT PROTECTION . . . . .	18
13.3.6	FreeMASTER . . . . .	21
13.3.7	FreeRTOS . . . . .	23
13.3.8	ADC Low Power . . . . .	25
13.3.9	AMMCLib . . . . .	27
13.3.10	LIN MASTER . . . . .	31
13.3.11	LIN SLAVE . . . . .	33
13.3.12	Structural Core Self Test Example . . . . .	35

13.4 Driver Examples . . . . .	36
13.4.1 Analog Driver Examples . . . . .	36
13.4.2 ADC Hardware Trigger . . . . .	37
13.4.3 ADC PAL example . . . . .	39
13.4.4 ADC Software Trigger . . . . .	42
13.4.5 CMP DAC . . . . .	43
13.4.6 Communication Driver Examples . . . . .	45
13.4.7 FLEXIO SPI . . . . .	46
13.4.8 LPUART Echo . . . . .	48
13.4.9 UART PAL ECHO . . . . .	50
13.4.10 CAN PAL . . . . .	51
13.4.11 LPSPI Transfer . . . . .	53
13.4.12 LPSPI DMA . . . . .	55
13.4.13 SPI PAL . . . . .	57
13.4.14 I2C PAL . . . . .	58
13.4.15 I2S PAL MASTER . . . . .	60
13.4.16 I2S PAL SLAVE . . . . .	62
13.4.17 FLEXIO I2C . . . . .	64
13.4.18 FLEXIO I2S MASTER . . . . .	65
13.4.19 FLEXIO I2S SLAVE . . . . .	67
13.4.20 FLEXIO UART . . . . .	69
13.4.21 LPI2C MASTER . . . . .	71
13.4.22 LPI2C SLAVE . . . . .	73
13.4.23 LIN MASTER BAREMETAL . . . . .	74
13.4.24 LIN SLAVE BAREMETAL . . . . .	76
13.4.25 SBC UJA116x . . . . .	78
13.4.26 System Driver Examples . . . . .	80
13.4.27 CRC Checksum . . . . .	81
13.4.28 CSEc key configuration . . . . .	83
13.4.29 EDMA transfer . . . . .	86
13.4.30 EIM INJECTION . . . . .	88
13.4.31 ERM REPORT . . . . .	90
13.4.32 EWM Interrupt . . . . .	93
13.4.33 FLASH Partitioning . . . . .	94
13.4.34 MPU Memory Protect Unit . . . . .	96
13.4.35 MPU PAL Memory Protection . . . . .	99
13.4.36 Power Mode Switch . . . . .	101
13.4.37 Trigger MUX Control . . . . .	104
13.4.38 WDG PAL Interrupt . . . . .	105
13.4.39 WDOG Interrupt . . . . .	107

13.4.40 SECURITY PAL . . . . .	108
13.4.41 Timer Driver Examples . . . . .	110
13.4.42 FTM Combined PWM . . . . .	111
13.4.43 FTM Periodic Interrupt . . . . .	112
13.4.44 FTM PWM . . . . .	114
13.4.45 FTM Signal Measurement . . . . .	116
13.4.46 IC PAL . . . . .	118
13.4.47 LPTMR Periodic Interrupt . . . . .	121
13.4.48 LPTMR Pulse Counter . . . . .	123
13.4.49 PDB Periodic Interrupt . . . . .	124
13.4.50 RTC Alarm . . . . .	126
13.4.51 TIMING PAL . . . . .	128
13.4.52 PWM PAL . . . . .	129
13.4.53 OC PAL . . . . .	131
13.4.54 LPIT Periodic Interrupt . . . . .	132
<b>14 Module Index</b>	<b>133</b>
14.1 Modules . . . . .	133
<b>15 Data Structure Index</b>	<b>138</b>
15.1 Data Structures . . . . .	138
<b>16 Module Documentation</b>	<b>139</b>
16.1 00. Library internals . . . . .	139
16.1.1 Detailed Description . . . . .	139
16.1.2 Internals . . . . .	139
16.1.3 Data Structure Documentation . . . . .	142
16.1.4 Enumeration Type Documentation . . . . .	148
16.1.5 Function Documentation . . . . .	148
16.1.6 Variable Documentation . . . . .	149
16.2 01. CAN & CAN partial networking configuration . . . . .	150
16.2.1 Detailed Description . . . . .	150
16.2.2 Introduction . . . . .	150
16.2.3 Data Structure Documentation . . . . .	151
16.2.4 Typedef Documentation . . . . .	154
16.2.5 Enumeration Type Documentation . . . . .	155
16.2.6 Function Documentation . . . . .	156
16.3 01. Mode handling . . . . .	159
16.3.1 Detailed Description . . . . .	159
16.3.2 Introduction . . . . .	159
16.3.3 Enumeration Type Documentation . . . . .	159



16.3.4	Function Documentation	160
16.4	01. Primary control	161
16.4.1	Detailed Description	161
16.5	01. Regulator configuration	162
16.5.1	Detailed Description	162
16.5.2	Introduction	162
16.5.3	Data Structure Documentation	162
16.5.4	Enumeration Type Documentation	163
16.5.5	Function Documentation	164
16.6	02. Battery monitor configuration	165
16.6.1	Detailed Description	165
16.6.2	Introduction	165
16.6.3	Data Structure Documentation	165
16.6.4	Typedef Documentation	167
16.6.5	Enumeration Type Documentation	167
16.6.6	Function Documentation	168
16.7	02. LIN configuration	169
16.7.1	Detailed Description	169
16.7.2	Introduction	169
16.7.3	Data Structure Documentation	169
16.7.4	Enumeration Type Documentation	170
16.7.5	Function Documentation	171
16.8	02. Supply control	173
16.8.1	Detailed Description	173
16.9	02. Watchdog	174
16.9.1	Detailed Description	174
16.9.2	Introduction	174
16.9.3	Data Structure Documentation	175
16.9.4	Enumeration Type Documentation	175
16.9.5	Function Documentation	176
16.1003.	Fail-safe configuration	178
16.10.1	Detailed Description	178
16.10.2	Introduction	178
16.10.3	Data Structure Documentation	179
16.10.4	Typedef Documentation	180
16.10.5	Enumeration Type Documentation	180
16.10.6	Function Documentation	181
16.1103.	SMPS configuration	182
16.11.1	Detailed Description	182
16.11.2	Introduction	182

16.11.3 Data Structure Documentation . . . . .	182
16.11.4 Typedef Documentation . . . . .	183
16.11.5 Enumeration Type Documentation . . . . .	183
16.11.6 Function Documentation . . . . .	184
16.1203. Transceiver control . . . . .	185
16.12.1 Detailed Description . . . . .	185
16.1304. HVIO bank 0 & 1 control . . . . .	186
16.13.1 Detailed Description . . . . .	186
16.13.2 Introduction . . . . .	186
16.13.3 Data Structure Documentation . . . . .	187
16.13.4 Typedef Documentation . . . . .	189
16.13.5 Enumeration Type Documentation . . . . .	189
16.13.6 Function Documentation . . . . .	190
16.1404. Lock control . . . . .	193
16.14.1 Detailed Description . . . . .	193
16.14.2 Introduction . . . . .	193
16.14.3 Data Structure Documentation . . . . .	193
16.14.4 Typedef Documentation . . . . .	195
16.14.5 Function Documentation . . . . .	195
16.1505. Timer control . . . . .	196
16.15.1 Detailed Description . . . . .	196
16.15.2 Introduction . . . . .	196
16.15.3 Data Structure Documentation . . . . .	196
16.15.4 Typedef Documentation . . . . .	197
16.15.5 Enumeration Type Documentation . . . . .	197
16.15.6 Function Documentation . . . . .	198
16.1606. Interrupts . . . . .	200
16.16.1 Detailed Description . . . . .	200
16.16.2 Introduction . . . . .	200
16.16.3 Function Documentation . . . . .	200
16.1707. MTPNV and ID . . . . .	204
16.17.1 Detailed Description . . . . .	204
16.17.2 Introduction . . . . .	204
16.17.3 Data Structure Documentation . . . . .	207
16.17.4 Function Documentation . . . . .	209
16.18ADC Driver . . . . .	211
16.18.1 Detailed Description . . . . .	211
16.18.2 Data Structure Documentation . . . . .	216
16.18.3 Enumeration Type Documentation . . . . .	219
16.18.4 Function Documentation . . . . .	223

16.19Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL)	229
16.19.1 Detailed Description	229
16.19.2 Data Structure Documentation	234
16.19.3 Typedef Documentation	237
16.19.4 Enumeration Type Documentation	237
16.19.5 Function Documentation	237
16.20Automotive Math and Motor Control Library	242
16.21Backward Compatibility Symbols for S32K142	243
16.22Bank N fail interrupt enable configuration options (0x38/0x48)	244
16.22.1 Detailed Description	244
16.22.2 Macro Definition Documentation	245
16.23Bank N wake-up interrupt enable configuration options (0x37/0x47)	247
16.23.1 Detailed Description	247
16.23.2 Macro Definition Documentation	248
16.24CRC Driver	250
16.24.1 Detailed Description	250
16.24.2 Data Structure Documentation	250
16.24.3 Enumeration Type Documentation	251
16.24.4 Function Documentation	251
16.25CSEc Driver	255
16.25.1 Detailed Description	255
16.25.2 Data Structure Documentation	260
16.25.3 Macro Definition Documentation	262
16.25.4 Typedef Documentation	263
16.25.5 Enumeration Type Documentation	263
16.25.6 Function Documentation	265
16.26Clock	276
16.26.1 Detailed Description	276
16.26.2 Function Documentation	276
16.27Clock Manager	277
16.27.1 Detailed Description	277
16.28Clock Manager Driver	278
16.28.1 Detailed Description	278
16.28.2 Data Structure Documentation	284
16.28.3 Macro Definition Documentation	302
16.28.4 Typedef Documentation	303
16.28.5 Enumeration Type Documentation	303
16.28.6 Function Documentation	310
16.28.7 Variable Documentation	315
16.29Common Core API	316

16.29.1 Detailed Description . . . . .	316
16.29.2 Macro Definition Documentation . . . . .	316
16.30 Common Transport Layer API . . . . .	318
16.30.1 Detailed Description . . . . .	318
16.30.2 Macro Definition Documentation . . . . .	318
16.30.3 Function Documentation . . . . .	321
16.31 Comparator (CMP) . . . . .	322
16.31.1 Detailed Description . . . . .	322
16.32 Comparator Driver . . . . .	326
16.32.1 Detailed Description . . . . .	326
16.32.2 Data Structure Documentation . . . . .	328
16.32.3 Macro Definition Documentation . . . . .	332
16.32.4 Typedef Documentation . . . . .	332
16.32.5 Enumeration Type Documentation . . . . .	332
16.32.6 Function Documentation . . . . .	335
16.33 Controller Area Network - Peripheral Abstraction Layer (CAN PAL) . . . . .	342
16.33.1 Detailed Description . . . . .	342
16.33.2 Data Structure Documentation . . . . .	347
16.33.3 Enumeration Type Documentation . . . . .	351
16.33.4 Function Documentation . . . . .	352
16.34 Controller Area Network with Flexible Data Rate (FlexCAN) . . . . .	359
16.34.1 Detailed Description . . . . .	359
16.35 Cooked API . . . . .	361
16.35.1 Detailed Description . . . . .	361
16.35.2 Function Documentation . . . . .	361
16.36 Cryptographic Services Engine (CSEc) . . . . .	363
16.36.1 Detailed Description . . . . .	363
16.37 Cyclic Redundancy Check (CRC) . . . . .	364
16.37.1 Detailed Description . . . . .	364
16.38 Diagnostic services . . . . .	366
16.38.1 Detailed Description . . . . .	366
16.38.2 Function Documentation . . . . .	367
16.39 Driver and cluster management . . . . .	370
16.39.1 Detailed Description . . . . .	370
16.39.2 Function Documentation . . . . .	370
16.40 EDMA Driver . . . . .	371
16.40.1 Detailed Description . . . . .	371
16.40.2 Data Structure Documentation . . . . .	376
16.40.3 Macro Definition Documentation . . . . .	383
16.40.4 Typedef Documentation . . . . .	383

16.40.5 Enumeration Type Documentation . . . . .	383
16.40.6 Function Documentation . . . . .	386
16.41 EIM Driver . . . . .	396
16.41.1 Detailed Description . . . . .	396
16.41.2 Data Structure Documentation . . . . .	398
16.41.3 Macro Definition Documentation . . . . .	398
16.41.4 Function Documentation . . . . .	399
16.42 ERM Driver . . . . .	401
16.42.1 Detailed Description . . . . .	401
16.42.2 ERM Driver Initialization . . . . .	401
16.42.3 ERM Driver Operation . . . . .	401
16.42.4 Data Structure Documentation . . . . .	403
16.42.5 Enumeration Type Documentation . . . . .	404
16.42.6 Function Documentation . . . . .	404
16.43 EWM Driver . . . . .	406
16.43.1 Detailed Description . . . . .	406
16.43.2 Data Structure Documentation . . . . .	408
16.43.3 Enumeration Type Documentation . . . . .	409
16.43.4 Function Documentation . . . . .	409
16.44 Enhanced Direct Memory Access (eDMA) . . . . .	411
16.44.1 Detailed Description . . . . .	411
16.45 Error Injection Module (EIM) . . . . .	412
16.45.1 Detailed Description . . . . .	412
16.46 Error Reporting Module (ERM) . . . . .	414
16.46.1 Detailed Description . . . . .	414
16.47 External Watchdog Monitor (EWM) . . . . .	416
16.47.1 Detailed Description . . . . .	416
16.48 Flash Memory (Flash) . . . . .	417
16.48.1 Detailed Description . . . . .	417
16.48.2 Data Structure Documentation . . . . .	420
16.48.3 Macro Definition Documentation . . . . .	421
16.48.4 Typedef Documentation . . . . .	425
16.48.5 Enumeration Type Documentation . . . . .	425
16.48.6 Function Documentation . . . . .	425
16.48.7 Variable Documentation . . . . .	434
16.49 Flash Memory (Flash) . . . . .	437
16.49.1 Detailed Description . . . . .	437
16.50 FlexCAN Driver . . . . .	440
16.50.1 Detailed Description . . . . .	440
16.50.2 Data Structure Documentation . . . . .	446

16.50.3 Typedef Documentation . . . . .	451
16.50.4 Enumeration Type Documentation . . . . .	451
16.50.5 Function Documentation . . . . .	454
16.51 FlexIO Common Driver . . . . .	462
16.51.1 Detailed Description . . . . .	462
16.51.2 Enumeration Type Documentation . . . . .	462
16.51.3 Function Documentation . . . . .	462
16.52 FlexIO I2C Driver . . . . .	465
16.52.1 Detailed Description . . . . .	465
16.52.2 Data Structure Documentation . . . . .	468
16.52.3 Function Documentation . . . . .	469
16.53 FlexIO I2S Driver . . . . .	474
16.53.1 Detailed Description . . . . .	474
16.53.2 Data Structure Documentation . . . . .	477
16.53.3 Typedef Documentation . . . . .	480
16.53.4 Function Documentation . . . . .	480
16.54 FlexIO SPI Driver . . . . .	492
16.54.1 Detailed Description . . . . .	492
16.54.2 Data Structure Documentation . . . . .	495
16.54.3 Typedef Documentation . . . . .	498
16.54.4 Enumeration Type Documentation . . . . .	499
16.54.5 Function Documentation . . . . .	499
16.55 FlexIO UART Driver . . . . .	506
16.55.1 Detailed Description . . . . .	506
16.55.2 Data Structure Documentation . . . . .	508
16.55.3 Enumeration Type Documentation . . . . .	510
16.55.4 Function Documentation . . . . .	510
16.56 FlexTimer (FTM) . . . . .	515
16.56.1 Detailed Description . . . . .	515
16.56.2 Data Structure Documentation . . . . .	521
16.56.3 Macro Definition Documentation . . . . .	525
16.56.4 Enumeration Type Documentation . . . . .	528
16.56.5 Function Documentation . . . . .	530
16.56.6 Variable Documentation . . . . .	552
16.57 FlexTimer Input Capture Driver (FTM_IC) . . . . .	553
16.57.1 Detailed Description . . . . .	553
16.57.2 Data Structure Documentation . . . . .	555
16.57.3 Enumeration Type Documentation . . . . .	557
16.57.4 Function Documentation . . . . .	558
16.58 FlexTimer Module Counter Driver (FTM_MC) . . . . .	561

16.58.1 Detailed Description . . . . .	561
16.58.2 Data Structure Documentation . . . . .	562
16.58.3 Function Documentation . . . . .	563
16.59 FlexTimer Output Compare Driver (FTM_OC) . . . . .	565
16.59.1 Detailed Description . . . . .	565
16.59.2 Data Structure Documentation . . . . .	567
16.59.3 Enumeration Type Documentation . . . . .	568
16.59.4 Function Documentation . . . . .	568
16.60 FlexTimer Pulse Width Modulation Driver (FTM_PWM) . . . . .	571
16.60.1 Detailed Description . . . . .	571
16.60.2 Data Structure Documentation . . . . .	578
16.60.3 Macro Definition Documentation . . . . .	583
16.60.4 Enumeration Type Documentation . . . . .	583
16.60.5 Function Documentation . . . . .	584
16.61 FlexTimer Quadrature Decoder Driver (FTM_QD) . . . . .	588
16.61.1 Detailed Description . . . . .	588
16.61.2 Data Structure Documentation . . . . .	590
16.61.3 Enumeration Type Documentation . . . . .	591
16.61.4 Function Documentation . . . . .	592
16.62 Flexible I/O (FlexIO) . . . . .	594
16.62.1 Detailed Description . . . . .	594
16.63 FreeRTOS . . . . .	595
16.64 HVIO high-side configuration options (0x71) . . . . .	596
16.64.1 Detailed Description . . . . .	596
16.64.2 Macro Definition Documentation . . . . .	597
16.65 HVIO low-side configuration options (0x72) . . . . .	599
16.65.1 Detailed Description . . . . .	599
16.65.2 Macro Definition Documentation . . . . .	600
16.66 I2S - Peripheral Abstraction Layer (I2S PAL) . . . . .	602
16.66.1 Detailed Description . . . . .	602
16.66.2 Data Structure Documentation . . . . .	604
16.66.3 Enumeration Type Documentation . . . . .	606
16.66.4 Function Documentation . . . . .	606
16.67 Initialization . . . . .	610
16.67.1 Detailed Description . . . . .	610
16.67.2 Function Documentation . . . . .	610
16.68 Input Capture - Peripheral Abstraction Layer (IC PAL) . . . . .	611
16.68.1 Detailed Description . . . . .	611
16.68.2 Data Structure Documentation . . . . .	615
16.68.3 Enumeration Type Documentation . . . . .	617

16.68.4 Function Documentation . . . . .	617
16.69 Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL) . . . . .	620
16.69.1 Detailed Description . . . . .	620
16.69.2 Data Structure Documentation . . . . .	624
16.69.3 Enumeration Type Documentation . . . . .	627
16.69.4 Function Documentation . . . . .	627
16.70 Interface management . . . . .	635
16.70.1 Detailed Description . . . . .	635
16.70.2 Function Documentation . . . . .	635
16.71 Interrupt Manager (Interrupt) . . . . .	637
16.71.1 Detailed Description . . . . .	637
16.71.2 Typedef Documentation . . . . .	638
16.71.3 Function Documentation . . . . .	638
16.72 Interrupt vector numbers for S32K142 . . . . .	642
16.73 J2602 Specific API . . . . .	643
16.74 J2602 Transport Layer specific API . . . . .	644
16.74.1 Detailed Description . . . . .	644
16.75 LIN 2.1 Specific API . . . . .	645
16.75.1 Detailed Description . . . . .	645
16.75.2 Function Documentation . . . . .	645
16.76 LIN Core API . . . . .	647
16.76.1 Detailed Description . . . . .	647
16.77 LIN Driver . . . . .	648
16.77.1 Detailed Description . . . . .	648
16.77.2 LIN Driver Overview . . . . .	648
16.77.3 LIN Driver Device structures . . . . .	648
16.77.4 LIN Driver Initialization . . . . .	649
16.77.5 LIN Data Transfers . . . . .	650
16.77.6 Autobaud feature . . . . .	650
16.77.7 Data Structure Documentation . . . . .	653
16.77.8 Macro Definition Documentation . . . . .	657
16.77.9 Typedef Documentation . . . . .	657
16.77.10 Enumeration Type Documentation . . . . .	657
16.77.11 Function Documentation . . . . .	658
16.77.12 Variable Documentation . . . . .	666
16.78 LIN Stack . . . . .	667
16.78.1 Detailed Description . . . . .	667
16.79 LPI2C Driver . . . . .	670
16.79.1 Detailed Description . . . . .	670
16.79.2 Data Structure Documentation . . . . .	673



16.79.3 Enumeration Type Documentation . . . . .	676
16.79.4 Function Documentation . . . . .	676
16.80LPIT Driver . . . . .	686
16.80.1 Detailed Description . . . . .	686
16.80.2 Data Structure Documentation . . . . .	690
16.80.3 Macro Definition Documentation . . . . .	691
16.80.4 Enumeration Type Documentation . . . . .	692
16.80.5 Function Documentation . . . . .	692
16.81LPSPI Driver . . . . .	701
16.81.1 Detailed Description . . . . .	701
16.81.2 Data Structure Documentation . . . . .	703
16.81.3 Enumeration Type Documentation . . . . .	709
16.81.4 Function Documentation . . . . .	710
16.81.5 Variable Documentation . . . . .	718
16.82LPTMR Driver . . . . .	719
16.82.1 Detailed Description . . . . .	719
16.82.2 Data Structure Documentation . . . . .	722
16.82.3 Enumeration Type Documentation . . . . .	723
16.82.4 Function Documentation . . . . .	725
16.83LPUART Driver . . . . .	729
16.83.1 Detailed Description . . . . .	729
16.83.2 Data Structure Documentation . . . . .	732
16.83.3 Enumeration Type Documentation . . . . .	736
16.83.4 Function Documentation . . . . .	736
16.84Local Interconnect Network (LIN) . . . . .	744
16.84.1 Detailed Description . . . . .	744
16.85Low Power Inter-Integrated Circuit (LPI2C) . . . . .	745
16.85.1 Detailed Description . . . . .	745
16.86Low Power Interrupt Timer (LPIT) . . . . .	746
16.86.1 Detailed Description . . . . .	746
16.87Low Power Serial Peripheral Interface (LPSPI) . . . . .	747
16.87.1 Detailed Description . . . . .	747
16.88Low Power Timer (LPTMR) . . . . .	750
16.88.1 Detailed Description . . . . .	750
16.89Low Power Universal Asynchronous Receiver-Transmitter (LPUART) . . . . .	751
16.89.1 Detailed Description . . . . .	751
16.90Low level API . . . . .	752
16.90.1 Detailed Description . . . . .	752
16.90.2 Data Structure Documentation . . . . .	755
16.90.3 Macro Definition Documentation . . . . .	771

16.90.4 Typedef Documentation . . . . .	773
16.90.5 Enumeration Type Documentation . . . . .	773
16.90.6 Function Documentation . . . . .	778
16.90.7 Variable Documentation . . . . .	782
16.91 MPU Driver . . . . .	784
16.91.1 Detailed Description . . . . .	784
16.91.2 Data Structure Documentation . . . . .	789
16.91.3 Enumeration Type Documentation . . . . .	791
16.91.4 Function Documentation . . . . .	795
16.92 MPU PAL . . . . .	798
16.92.1 Detailed Description . . . . .	798
16.92.2 Data Structure Documentation . . . . .	801
16.92.3 Typedef Documentation . . . . .	803
16.92.4 Enumeration Type Documentation . . . . .	805
16.92.5 Function Documentation . . . . .	806
16.93 Memory Protection Unit (MPU) . . . . .	809
16.93.1 Detailed Description . . . . .	809
16.94 Memory Protection Unit Peripheral Abstraction Layer (MPU PAL) . . . . .	811
16.94.1 Detailed Description . . . . .	811
16.95 Node configuration . . . . .	816
16.95.1 Detailed Description . . . . .	816
16.95.2 Function Documentation . . . . .	816
16.96 Node configuration . . . . .	818
16.96.1 Detailed Description . . . . .	818
16.96.2 Function Documentation . . . . .	818
16.97 Node identification . . . . .	823
16.97.1 Detailed Description . . . . .	823
16.97.2 Function Documentation . . . . .	823
16.98 Notification . . . . .	824
16.99 OS Interface (OSIF) . . . . .	825
16.99.1 Detailed Description . . . . .	825
16.99.2 Macro Definition Documentation . . . . .	827
16.99.3 Function Documentation . . . . .	827
16.100 Output Compare - Peripheral Abstraction Layer (OC PAL) . . . . .	834
16.100.1 Detailed Description . . . . .	834
16.100.2 Data Structure Documentation . . . . .	837
16.100.3 Enumeration Type Documentation . . . . .	839
16.100.4 Function Documentation . . . . .	840
16.101 PDB Driver . . . . .	845
16.101.1 Detailed Description . . . . .	845

16.101.2	Data Structure Documentation	849
16.101.3	Enumeration Type Documentation	850
16.101.4	Function Documentation	851
16.102	PINS Driver	857
16.102.1	Detailed Description	857
16.102.2	Data Structure Documentation	857
16.102.3	Typedef Documentation	858
16.102.4	Enumeration Type Documentation	858
16.102.5	Function Documentation	859
16.103	Peripheral access layer for S32K142	862
16.104	Pins Driver (PINS)	863
16.104.1	Detailed Description	863
16.105	Power Manager	865
16.105.1	Detailed Description	865
16.105.2	Data Structure Documentation	866
16.105.3	Typedef Documentation	868
16.105.4	Enumeration Type Documentation	869
16.105.5	Function Documentation	870
16.105.6	Variable Documentation	874
16.106	Power Manager Driver	875
16.107	Power_s32k1xx	877
16.107.1	Detailed Description	877
16.107.2	Data Structure Documentation	878
16.107.3	Enumeration Type Documentation	879
16.107.4	Function Documentation	881
16.108	Programmable Delay Block (PDB)	883
16.108.1	Detailed Description	883
16.109	Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL)	884
16.109.1	Detailed Description	884
16.109.2	Data Structure Documentation	887
16.109.3	Enumeration Type Documentation	889
16.109.4	Function Documentation	889
16.110	RTC Driver	893
16.110.1	Detailed Description	893
16.110.2	Data Structure Documentation	895
16.110.3	Macro Definition Documentation	899
16.110.4	Enumeration Type Documentation	900
16.110.5	Function Documentation	901
16.111	Raw API	908
16.111.1	Detailed Description	908

16.111.2	Function Documentation	908
16.112	Real Time Clock Driver (RTC)	910
16.112.1	Detailed Description	910
16.113	Register address map	914
16.113.1	Detailed Description	914
16.113.2	Macro Definition Documentation	918
16.114	S32K142 SoC Header file	927
16.114.1	Detailed Description	927
16.115	S32K142 System Files	928
16.116	BC configuration control options (0x74)	929
16.116.1	Detailed Description	929
16.116.2	Macro Definition Documentation	930
16.117	Schedule management	932
16.117.1	Detailed Description	932
16.117.2	Function Documentation	932
16.118	Security PAL	933
16.118.1	Detailed Description	933
16.118.2	Data Structure Documentation	935
16.118.3	Enumeration Type Documentation	935
16.118.4	Function Documentation	937
16.119	Security Peripheral Abstraction Layer - SECURITY PAL	951
16.119.1	Detailed Description	951
16.120	Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL)	954
16.120.1	Detailed Description	954
16.120.2	Data Structure Documentation	957
16.120.3	Enumeration Type Documentation	960
16.120.4	Function Documentation	961
16.121	Signal interaction	966
16.122	SoC Header file (SoC Header )	967
16.122.1	Detailed Description	967
16.123	SoC Support	968
16.123.1	Detailed Description	968
16.124	Start-up control configuration options (0x73)	970
16.124.1	Detailed Description	970
16.124.2	Macro Definition Documentation	971
16.125	Structural Core Self Test	973
16.126	Supply interrupt enable configuration options (0x1C)	975
16.126.1	Detailed Description	975
16.126.2	Macro Definition Documentation	976
16.127	System Basis Chip Driver (SBC) - UJA116xA Family	978

16.127.1	Detailed Description	978
16.128	System Basis Chip library (SBC) - UJA113x Family	983
16.128.1	Detailed Description	983
16.128.2	Introduction	983
16.129	System interrupt enable configuration options (0x04)	984
16.129.1	Detailed Description	984
16.129.2	Macro Definition Documentation	984
16.130	IRGMUX Driver	985
16.130.1	Detailed Description	985
16.130.2	Data Structure Documentation	986
16.130.3	Typedef Documentation	987
16.130.4	Function Documentation	988
16.131	Timing - Peripheral Abstraction Layer (TIMING PAL)	992
16.131.1	Detailed Description	992
16.131.2	Data Structure Documentation	996
16.131.3	Enumeration Type Documentation	999
16.131.4	Function Documentation	999
16.132	Transceiver interrupt enable configuration options (0x23)	1003
16.132.1	Detailed Description	1003
16.132.2	Macro Definition Documentation	1004
16.133	Transport layer API	1005
16.133.1	Detailed Description	1005
16.134	UJA116xA SBC Driver	1006
16.134.1	Detailed Description	1006
16.134.2	Data Structure Documentation	1013
16.134.3	Macro Definition Documentation	1029
16.134.4	Typedef Documentation	1029
16.134.5	Enumeration Type Documentation	1030
16.135	Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL)	1047
16.135.1	Detailed Description	1047
16.135.2	Data Structure Documentation	1052
16.135.3	Enumeration Type Documentation	1053
16.135.4	Function Documentation	1054
16.136	User provided call-outs	1060
16.136.1	Detailed Description	1060
16.136.2	Function Documentation	1060
16.137	WDG PAL	1061
16.137.1	Detailed Description	1061
16.137.2	Data Structure Documentation	1062
16.137.3	Enumeration Type Documentation	1063

16.137.4	Function Documentation	1064
16.138	WDOG Driver	1068
16.138.1	Detailed Description	1068
16.138.2	Data Structure Documentation	1071
16.138.3	Enumeration Type Documentation	1072
16.138.4	Function Documentation	1073
16.139	Watchdog Peripheral Abstraction Layer (WDG PAL)	1077
16.139.1	Detailed Description	1077
16.140	Watchdog timer (WDOG)	1080
16.140.1	Detailed Description	1080
<b>17</b>	<b>Data Structure Documentation</b>	<b>1081</b>
17.1	adc_callback_info_t Struct Reference	1081
17.1.1	Detailed Description	1081
17.1.2	Field Documentation	1081
17.2	adc_instance_t Struct Reference	1081
17.2.1	Detailed Description	1081
17.2.2	Field Documentation	1082
17.3	can_instance_t Struct Reference	1082
17.3.1	Detailed Description	1082
17.3.2	Field Documentation	1082
17.4	drv_config_t Struct Reference	1082
17.4.1	Detailed Description	1083
17.4.2	Field Documentation	1083
17.5	i2c_instance_t Struct Reference	1083
17.5.1	Detailed Description	1083
17.5.2	Field Documentation	1083
17.6	i2s_instance_t Struct Reference	1084
17.6.1	Detailed Description	1084
17.6.2	Field Documentation	1084
17.7	ic_instance_t Struct Reference	1084
17.7.1	Detailed Description	1084
17.7.2	Field Documentation	1085
17.8	lin_product_id_t Struct Reference	1085
17.8.1	Detailed Description	1085
17.8.2	Field Documentation	1085
17.9	mpu_instance_t Struct Reference	1086
17.9.1	Detailed Description	1086
17.9.2	Field Documentation	1086
17.10	oc_instance_t Struct Reference	1086

17.10.1 Detailed Description	1086
17.10.2 Field Documentation	1087
17.11 oc_pal_state_t Struct Reference	1087
17.11.1 Detailed Description	1087
17.12 pwm_instance_t Struct Reference	1087
17.12.1 Detailed Description	1087
17.12.2 Field Documentation	1087
17.13 spi_instance_t Struct Reference	1088
17.13.1 Detailed Description	1088
17.13.2 Field Documentation	1088
17.14 timer_chan_state_t Struct Reference	1088
17.14.1 Detailed Description	1089
17.15 timing_instance_t Struct Reference	1089
17.15.1 Detailed Description	1089
17.15.2 Field Documentation	1089
17.16 uart_instance_t Struct Reference	1089
17.16.1 Detailed Description	1090
17.16.2 Field Documentation	1090
17.17 wdg_instance_t Struct Reference	1090
17.17.1 Detailed Description	1090
17.17.2 Field Documentation	1090
<b>Index</b>	<b>1093</b>

## 1 S32 SDK

### Introduction

This topic provides an introduction to the S32 software development kit (S32 SDK), including intended audience, purpose and scope, and detailed sections on technical considerations.



Copyright © 2016 NXP Semiconductor



### Intended Audience

S32 SDK documentation is written for software developers and system engineers who have a technical background, and a working knowledge of embedded programming. The audience for the S32 SDK are users of S32 Processors.

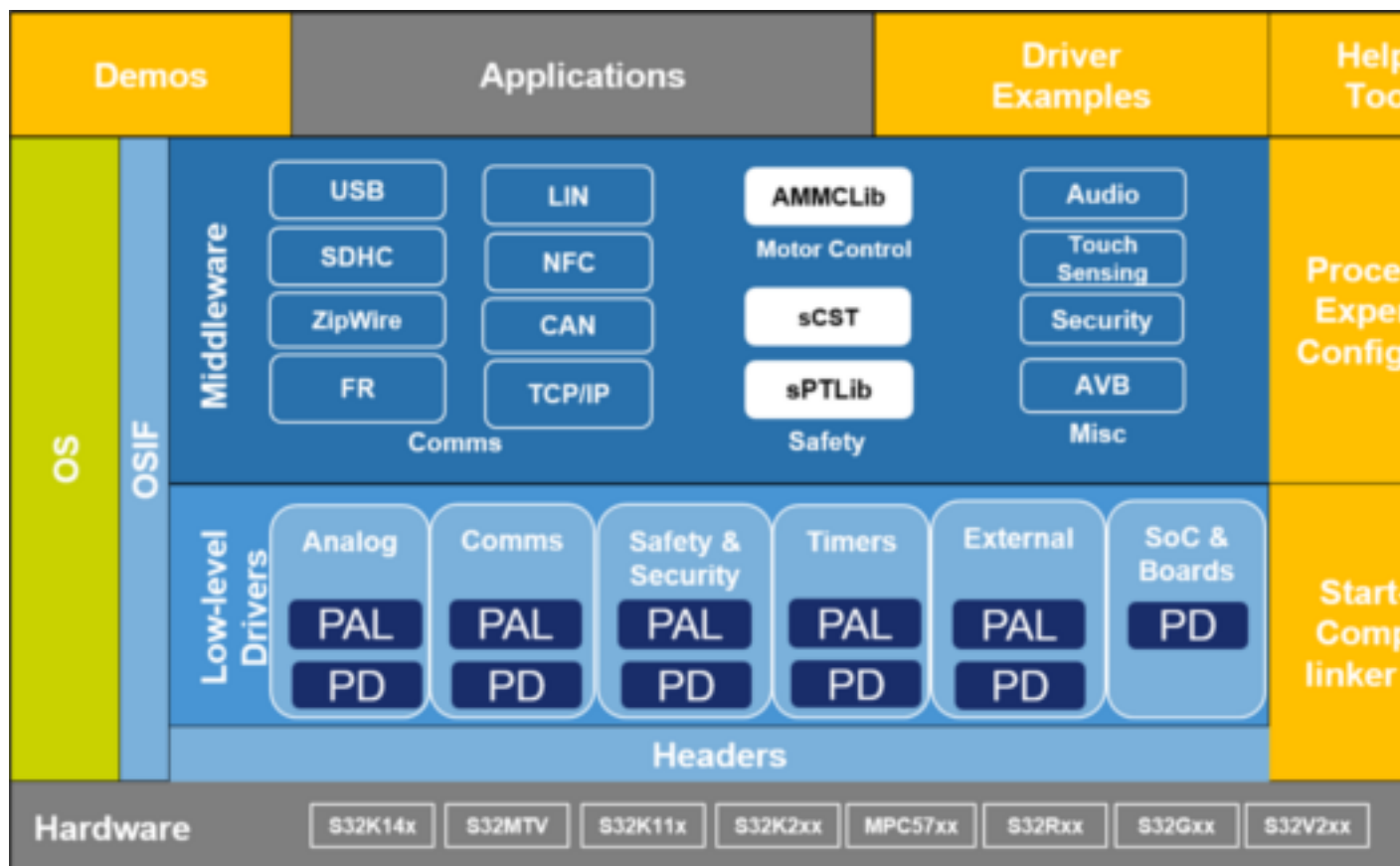
### Purpose and Scope

The S32 SDK is an embedded oriented development kit. It allows users to

1. Evaluate and explore the features of the S32 processors; experience how they are supported by working "out of the box" on NXP development boards.
2. Develop embedded solutions; the NXP SDK is thoroughly tested from development to production.

### S32 SDK Architecture Overview

The S32 SDK is an extensive suite of robust hardware interface and hardware abstraction layers, peripheral drivers, RTOS, stacks, and middleware designed to simplify and accelerate application development on NXP S32 SOC's. The addition of Processor Expert technology for software and board configuration provides unmatched ease of use and flexibility. Included in the S32 SDK is full source code under a permissive open-source license for all hardware abstraction and peripheral driver software. See the Release Notes for details. The S32 SDK consists of the following runtime software components written in C:



## 2 Components



### Header file

The S32 SDK contains a device-specific header files which provide direct access to the peripheral registers. Each supported device in S32 SDK has an overall System-on-Chip (SoC) memory-mapped header file. This header file contains the memory map and register base address for each peripheral and the IRQ vector table with associated vector numbers.

### Feature Header File

The PAL is designed to be reusable regardless of the peripheral configuration differences from one SOC device to another. An overall Peripheral Feature Header File is provided for device to define the feature or configuration differences for each SOC sub-family device.

### Peripheral Abstraction Layer

The PAL provides unified interfaces for families of peripherals, allowing for cross-platform compatibility of application code. The main goal is to provide an application programming interface that is independent of the underlying peripheral implementation.

The PAL supports all instances of each peripheral from a certain family instantiated on the SOC by using a simple integer parameter for the peripheral instance number.

The PAL instances should be configured bearing in mind possible limitations of the underlying peripherals - some features may not be supported on some hardware modules. It is the user's responsibility to correctly handle hardware resources, especially when porting the application to a different platform.

The PAL drivers can be found in the platform/pal directory.

### Peripheral Drivers

The Peripheral Drivers are high-level drivers that implement high-level logic transactions based on an internal register access abstraction layer, other Peripheral Drivers, and/or System Services. For example, the UART register access abstraction layer 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. In general, if a driver, that is mainly based on one peripheral, interfaces with functions beyond the register access abstraction layer and/or requires interrupt servicing, the driver is considered a high-level Peripheral Driver.

The Peripheral Drivers support all instances of each peripheral instantiated on the SOC by using a simple integer parameter for the peripheral instance number. The user of the Peripheral Driver does not need to know the peripheral memory-mapped base address.

The Peripheral Drivers operate on a high-level logic that requires data storage for internal operation context handling. However, the Peripheral Drivers do not allocate this memory space. Rather, the user passes in the memory for the driver internal operation through the driver initialization function.

The Peripheral Drivers are designed to handle the entire functionality for a targeted use-case. An application should be able to use only the Peripheral Driver to accomplish its purpose.

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

### System Services

The System Services contain a set of software entities that can be used by the Peripheral Drivers. They may be used with PAL 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 platform/drivers 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 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.

### 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 components, 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. The Power Manager provides a notification framework which the software components, such as drivers, uses to register callback functions and execute the predefined code flow during the power mode transition.

### Examples

The examples provided show how to build user applications using the S32 SDK. The examples can be found in the top-level example directory. For details please see [Examples\\_and\\_Demos](#).

## 3 PAL vs PD usage

### PAL - Peripheral Abstraction layer

- Interface abstraction for a family of peripherals (E.g. LPUART + LINFlexD\_UART + eSCI + FlexIO\_UART + etc.)
- Single layer per SDK
- Same generic API on multiple platforms

### PD - Peripheral Drivers

- IP dedicated low-level drivers
- Designed for efficiency and IP features set coverage

### When to use the Peripheral Abstraction Layer (PAL)?

- Whenever an application needs a simplified, generic interface that abstracts as much as possible the underlying silicon features.
- Whenever developing portable higher level generic code that is meant to run on different NXP platforms. This may include anything from low level console utility libraries to communication stacks like TCP/IP.

### When to use Peripheral Drivers?

- Whenever developing for high efficiency (code size, execution speed, etc.) or planning to use specific peripheral features.

## 4 Supported Platforms

Supported board and SoC versions can be found in the Release Notes. (SDK\ReleaseNotes.pdf)

## 5 Installation

### Prerequisites

SDK can be used in two ways: bundled in S32 Design Studio and standalone.

S32 SDK is delivered bundled in the S32 Design Studio. In this case it's already configured and ready to use.

S32 SDK is also delivered through a standalone installer. Using the standalone installer is recommended when using a compiler which is not supported in S32 Design Studio or when the graphical interface is not required. In this case the installer can configure an existing S32 Design Studio to use the configuration files delivered in the installer.

If the integration with the S32 Design Studio is not needed the path to S32 Design Studio can be left empty – and in this case only the S32 SDK will be installed and configured.

### Steps

1. Start the installer S32\_SDK\_<ReleaseSpecifc>.exe
2. Set the destination folder for the SDK, give optional location of S32DS and install. Example of S32DS path: C:\NXP\S32ARMv1.3
3. Start using the SDK by creating a new project or importing a project

### Background

The installer does the following things in background:

- Puts the SDK in the selected destination directory.
- Appends to S32SDK\_PATH the path of the SDK.
  - Note: Please make sure you uninstall previous SDK so that this variable will be empty.
- Copies necessary files into S32 Design Studio installation location.
- Overwrites existing SDK from S32 Design Studio with the version from destination directory

### Uninstaller

When the SDK is installed using the standalone the installer, the user can use "uninst.exe" from the root of the destination to uninstall the SDK.

Note: If you want to reinstall the SDK please use a clean copy of S32DS. When you uninstall this does not delete the copied files (ex: Config\_01.pez), so a clean copy is needed.

## 6 Build Tools

### Introduction

S32 SDK supports and is tested with multiple compiler toolchains.

**Note**

The toolchain list, versions and their options specific for the platform and release can be found in the Release Notes. (SDK\ReleaseNotes.pdf)

Toolchain versions and options can be found in the Release Notes. (SDK\ReleaseNotes.pdf)

**Compiler warnings disabled for S32 SDK**

For *Wind River DIAB Compiler* the following warnings are not checked at S32 SDK build time:

- *#1824: explicit cast discards volatile qualifier*  
Motivation: this warning has been deactivated because of false positive occurrences reported for Wind River DIAB Compiler 5.9.4.8 under tickets TCDIAB-13994, TCDIAB-14098.
- *#5387: explicit conversion of a 64-bit integral type to a smaller integral type (potential portability problem)*  
Motivation: this warning has been disabled because it is reported for conversions required by the internal SDK algorithms. Intermediary results requiring high precision are stored as uint64\_t variables and converted into uint32\_t variables. Checks have been put in place to ensure that the cast is only done if the value to be converted fits on 32 bits.
- *#5388: conversion from pointer to same-sized integral type (potential portability problem)*  
Motivation: for S32 SDK conversions between uint32\_t and memory addresses are made assuming that pointers are stored on 32bits.

**Makefiles**

Multiple makefile projects are provided in the 'examples' folder, for all supported compilers. These projects can be modified by adding application code, or the makefiles can be reused in different projects, after reconfiguring the paths/variables. Please note that these projects require the designated compiler to be already installed on the host; also, the makefile path to compiler executable must be updated before running make utility.

**S32 Design Studio**

S32 Design Studio is delivered with platform specific gcc cross compiler included ("S32\_Design\_Studio\_install\_↔ path\Cross\_Tools). Eclipse plugins for gcc are already installed in S32 Design Studio IDE, so new projects for this toolchain can be created and built directly from the IDE. To add S32 SDK source files to a clean S32 Design Studio project, eclipse "linked resources" feature can be used: project properties->New->Folder->Advanced->'Link to alternate location' (e.g. "S32\_SDK\_PATH"). For S32 Design Studio project with Processor Expert support, please import a project from "S32\_SDK\_PATH Name".

## 7 IDE Support

**S32 Design Studio**

- S32 Design Studio is delivered with Processor Expert support included. Please see [Configuration](#) chapter.
- To configure the S32 SDK path of the project, eclipse "S32 SDK Specific" feature can be used: patch project properties->Processor Expert->S32 SDK Specific->SDK path
- Processor Expert repositories and paths can be configured as it follows: Window -> Preferences -> Processor Expert -> Repositories and Paths.
- S32 Design Studio projects can be imported from S32 SDK package. Please see [Examples\\_and\\_Demos](#) chapter.

### IAR Embedded Workbench

- NOT applicable to platforms which do not support IAR compiler. Please see Release Notes.
- There is no configuration support for S32 SDK in IAR.
- IAR Embedded Workbench projects can be imported from S32 SDK package. Please see Examples\_and\_Demos chapter.

## 8 Configuration

Processor Expert software allows generation of configuration structures for peripheral drivers from S32 SDK. With the help of Eclipse based graphical interface where you can configure your driver and generate corresponding configuration structure. This tool doesn't generate source code for S32 family, it only generates configurations data structures.

Processor Expert generates configuration header files that are included by application source code. The configuration data structures from these files are defined in S32 SDK. All these header files are generated by this tool in \${ProjName}/Generated\_Code directory.

Peripheral drivers are not stored directly in the project directory, these drivers are stored in S32 SDK repository. Shared peripheral drivers repository is advantageous when more projects should share the same version of peripheral drivers. In this case, peripheral drivers are not physically placed in the project directory but each project is virtually linked with shared, common repository from S32 SDK. This way the management of the projects' drivers can be done in one place and any changes made in the shared repository is automatically distributed across all of the linked projects, for example in case of bug fixing or library update and also backup or archiving of the peripheral drivers versions is very simple.

## 9 Acronyms and Abbreviations

Acronym	Description
CPSIE, CPSID	Change Processor State Interrupt Enable / Disable
EAR	Early Access Release
EVB	Evaluation board
PAL	Peripheral Abstraction Layer
IRQ	Interrupt Request
ISR	Interrupt Service Routine
LLWU	Low Leakage Wakeup Unit
NVIC	Nested Vector Interrupt Controller
RTOS	Real Time Operating System
S32DS	S32 Design Studio
SDK	Software Development Kit
SOC	System-on-Chip
UART	Universal Asynchronous Receiver / Transmitter

## 10 MISRA Compliance

This section describes how the S32 SDK project addresses MISRA Compliance.

The S32 SDK SW components which are implemented to be compliant with MISRA C 2012 are:

- all drivers & PALs
- generated driver code (including Cpu.c & .h)
- main.c (generated via graphical configurator)

Violations of MISRA C 2012 guidelines which remain not fixed, shall be documented as deviations at file level.

Other SW components included in the S32 SDK package which are not subject to MISRA C 2012 compliance:

- demo\_apps & driver examples
- FreeRTOS

## 11 Development guidelines

Set of guidelines to improve the usability of the S32 SDK.

Some usual guidelines on SDK programming model:

1. Driver state structures should be declared as global or static variables as they are used in the whole time when the driver is used.
2. Driver state structures content should not be used or modified by the application code.
3. Peripheral drivers, PALs and Middleware code are not handling clock and pins initialization. Configuration of the clock and pins driver has to be done by the application. To make sure these are properly initialized before other modules are used, please call the corresponding initialization:

```
/* Initialize and configure clocks */
CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
               g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
CLOCK_SYS_UpdateConfiguration(0U,
                              CLOCK_MANAGER_POLICY_AGREEMENT);

/* Initialize pins */
PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);
```

### Note

The configuration structure names used in this example are the default names generated by Processor Expert components for clock and pins. Applications not using Processor Expert might have different names for these structures.

4. The recommended approach at development time is to add DEV\_ERROR\_DETECT symbol to the compiler defines. This will enable DEV\_ASSERT mechanism which can catch application code errors in the early development stage.
5. High care should be taken to have a backup option when debug pins are routed to other functionalities.

## 12 Error detection and reporting

S32 SDK drivers can use a mechanism to validate data coming from upper software layers (application code) by performing a number of checks on input parameters' range or other invariants that can be statically checked (not dependent on runtime conditions). A failed validation is indicative of a software bug in application code, therefore it is important to use this mechanism during development.

The validation is performed by using DEV\_ASSERT macro. A default implementation of this macro is provided in this file. However, application developers can provide their own implementation in a custom file. This requires defining the CUSTOM\_DEVASSERT symbol with the specific file name in the project configuration (for example: -DCUSTOM\_DEVASSERT="custom\_devassert.h")

The default implementation accommodates two behaviors, based on DEV\_ERROR\_DETECT symbol:

- When DEV\_ERROR\_DETECT symbol is defined in the project configuration (for example: -DDEV\_ERROR\_DETECT), the validation performed by the DEV\_ASSERT macro is enabled, and a failed validation triggers a software breakpoint and further execution is prevented (application spins in an infinite loop) This configuration is recommended for development environments, as it prevents further execution and allows investigating potential problems from the point of error detection.

- When `DEV_ERROR_DETECT` symbol is not defined, the `DEV_ASSERT` macro is implemented as no-op, therefore disabling all validations. This configuration can be used to eliminate the overhead of development-time checks.

It is the application developer's responsibility to decide the error detection strategy for production code: one can opt to disable development-time checking altogether (by not defining `DEV_ERROR_DETECT` symbol), or one can opt to keep the checks in place and implement a recovery mechanism in case of a failed validation, by defining `CUSTOM_DEVASSERT` to point to the file containing the custom implementation.

## 13 Examples and Demos

Applications that show the user how to initialize the peripherals for the basic use cases

### 13.1 Introduction

S32 SDK examples structure:

- [Demo Applications](#) (SDK/examples/<CPU>/demo\_apps), are demo applications for various IDEs and compilers. Also this examples are using more advanced use-cases - FreeRTOS integration, LIN Stack, FlexCAN usage and Clock Setup.
- [Driver Examples](#) (SDK/examples/<CPU>/driver\_examples), are simple applications which exemplify a basic use-case for a specific driver.

### 13.2 Usage

#### 13.2.1 How to build

For makefile project

There are makefile projects in all compilers supported. In order to used them:

- **Make** utility (eg. GNU Make)
- **Toolchain** (eg. GCC Toolchain)
- **Make sure the make and compiler are in Path (for Microsoft Windows : System -> Environmental Variables)**
- From command line execute the makefile: **make all**

The makefiles generate binary files for both RAM and FLASH configurations.

For IAR Embedded Workbench

From IAR Workbench for ARM use File > Open > Workspace and browse to the desired project. After the project was opened you can see the files in "Workspace Files". Finally, the project can be executed from Project > Download and Debug. Make sure that the debug probe you are using is selected and configured in Project options > Debugger > Driver.

For S32 Design Studio

From S32 Design Studio (See Release notes for the S32 Design Studio version), go to File -> New -> New Project from Example and select the example you wish to import. This will copy the example project into workspace. Next steps:

- Examples will run without an active configuration, however if any changes are required, a configuration needs to be generated. Use Open S32 Configuration button, make the desired changes (if any) then click on the "Update Code" button.
- Use Project > Build to build the project
- Use Project > Debug and launch your preferred debug configuration

### 13.2.2 How to debug

This section explains how to upload and debug the binary files generated after build. This assumes that you have a debug probe(see release notes for supported debug probes) and a debug software installed on the machine.

Generic steps:

1. Launch the debug software
2. Load the binary file into the MCU
3. Execute the application

#### Loading with Segger JLink:

- Download and install the latest drivers and GDB server, named *Software and documentation pack*, from their [site](#)
- Download your favorite GDB client (eg. arm-none-eabi-gdb)
- Browse to JLink installation folder and launch **JLinkGDBServer**
- Select the appropriate part from the device list and click on **OK**
- Open the GDB client and connect to the configured port - by default localhost:2331
- Upload the file and execute (see GDB client user manual for details regarding the commands used)

The following table is a small list of commands used in GNU ARM GDB with JLinkGDBServer to connect and run the application:

Command	Description
target remote:PortNumber	Connect to the remote target at a specified port. Please replace PortNumber with the port configured in the GDB server.
monitor reset	Reset the target MCU
monitor halt	Halt the target MCU
file ApplicationName.elf	Load the file and symbols. Please change ApplicationName with your application name
load	Download the executable to the target MCU
continue	Begin executing the application

#### Loading with PEmicro OpenSDA/MultiLink:

- Download and install the latest drivers and GDB server, named *P&E GDB Server for Kinetis with Windows GUI*, from their [site](#)
- Download your favorite GDB client (eg. arm-none-eabi-gdb)
- Browse to PEmicro GDB Server installation folder and launch **P&E GDB Server for Kinetis**
- Select the appropriate part from the device list and click on **Connect**
- Open the GDB client and connect to the configured port - by default localhost:7224
- Upload the file and execute (see GDB client user manual for details regarding the commands used)

The following table is a small list of commands used in GNU ARM GDB with PEmicro GDB server to connect and run the application:



Command	Description
target remote:PortNumber	Connect to the remote target at a specified port. Please replace PortNumber with the port configured in the GDB server.
monitor reset	Reset the target MCU
file ApplicationName.elf	Load the file and symbols. Please change ApplicationName with your application name
load	Download the executable to the target MCU
continue	Begin executing the application

### 13.2.3 Using terminal emulator

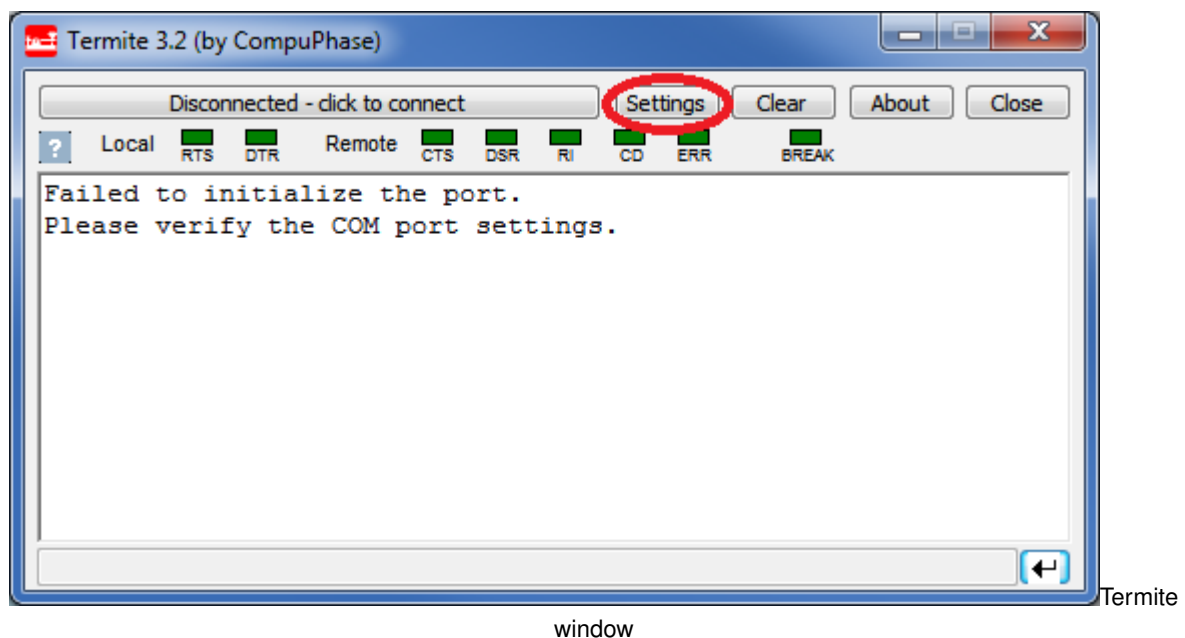
To run the examples that use LPUART to help you visualize data you must download a terminal emulator (eg. Putty, Termit, TeraTerm) and configure it.

Unless otherwise noted the standard communication parameters are:

- 115200 baud
- One stop bit
- No parity
- No flow control

Example configuration for Termit using OpenSDA

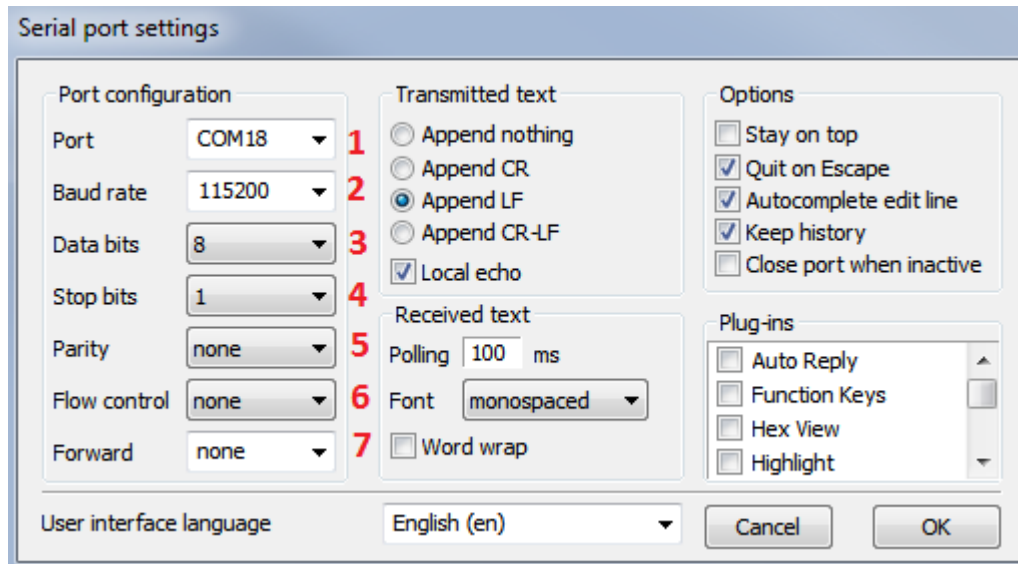
- 1) Download Termit from their [site](#)
- 2) Run the installer. Wait for the installation to be completed
- 3) Go to **Start -> All Programs -> Termit** and launch the program. The window from Fig.1 will appear ...



- 4) Click on **Settings**
- 5) As seen in Fig.2, configure the following communication parameters:

- **Port(1)** : COMx - where x must be replaced with the COM port number
- **Baud Rate(2)** : 115200

- **Data Bits(3)** : 8
- **Stop Bits(4)** : 1
- **Parity(5)** : None
- **Flow Control(6)** : None
- **Forward(7)** : None



Settings window

6) Click **OK**. Now the terminal should be configured

#### Note

For further help consult the terminal's documentation

## 13.3 Demo Applications

Applications that show more advanced use cases

Available demo applications:

*Click on one of the project to see the corresponding documentation*

- [Hello World](#)
- [Hello World - IAR Embedded Workbench](#)
- [Hello World - Makefile](#)
- [FlexCAN Encrypted](#)
- [CSEC BOOT PROTECTION](#)
- [FreeMASTER](#)

- [FreeRTOS](#)
- [ADC Low Power](#)
- [AMMCLib](#)
- [LIN MASTER](#)
- [LIN SLAVE](#)
- [Structural Core Self Test Example](#)

#### 13.3.1 Hello World

Basic application that presents the project scenarios for S32 SDK

##### Application description

The purpose of this demo is to provide the user with an out-of-the box example application for S32K142 platform, using S32 SDK. The demo uses hardware abstraction layer primitives for PCC and PORT modules in order to toggle two LEDs alternatively.

##### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- Debug probe (JLink, PEmicro, OpenSDA)

##### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

##### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
RED_LED (PTD15)	RGB_RED - wired on the board	J12.17 - J11.31

GREEN_LED (PTD16)	RGB_GREEN - wired on the board	J12.16 - J11.30
-------------------	--------------------------------	-----------------

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **hello\_world\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace. Wait for the S32 Configuration was initialized and ready.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **RAM** (Debug\_RAM) or **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
hello_world_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
hello_world_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
hello_world_s32k142_debug_ram_pemicro	Debug the RAM configuration using PEMicro debuggers
hello_world_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

## Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### 13.3.2 Hello World - IAR Embedded Workbench

Basic application that presents the project scenarios for S32 SDK and integration with IAR Embedded Workbench IDE

## Application description

The purpose of this demo is to provide the user with an out-of-the box example application for S32K142 platform, using S32 SDK. The demo uses Clock Manager and Pins Driver to toggle two LEDs alternately.

**Note**

For information about how to run IAR projects please refer to [Usage](#)  
The default debug probe is "Simulator". Prior to running the project on the board, the desired debug probe has to be configured in Project Options -> Debugger.

**Prerequisites**

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

**Boards supported**

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

**Hardware Wiring**

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
RED_LED (PTD15)	RGB_RED - wired on the board	J12.17 - J11.31
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	J12.16 - J11.30

**13.3.3 Hello World - Makefile**

Basic application that presents the project scenarios for S32 SDK using makefiles for various compilers

**Application description**

The purpose of this demo is to provide the user with an out-of-the box example application for S32K142 platform, using S32 SDK. The demo uses Pins and Clock driver to initialize the MCU and to toggle two LEDs alternatively.

There are five projects delivered with this package:

- Makefile project (GCC compiler)
- Makefile project (GHS compiler)
- Makefile project (IAR compiler)
- Makefile project (DCC compiler)
- Makefile project (ARM compiler)

**Note**

For information about how to run the makefile please refer to [Usage](#)

## Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- Debug probe (JLink, PEmicro, OpenSDA)

## Boards supported

The following boards are supported by this application:

- S32K142EVb-Q100
- S32K142-MB

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVb-Q100	S32K142-MB
LED1 (PTD15/PTC0)	RGB_RED - wired on the board	J12.17 - J11.31
LED2 (PTD16/PTC1)	RGB_GREEN - wired on the board	J12.16 - J11.30

### 13.3.4 FlexCAN Encrypted

Demo application showing the FlexCAN functionalities

#### Note

**If running the encrypted communication:** The encryption uses the first non-volatile user key, which needs to be configured by running the **CSEc Key Configuration** in the driver examples folder.

**Encrypted communication works only for CSEc enabled parts.** SIM\_SDID indicates whether CSEc is available on your device.

**If one of the user keys was loaded using the CSEc Key Configuration, any further full erase of the Flash requires a Challenge-Authentication process.** This can be done by running the CSEc Key Configuration example again and setting the ERASE\_ALL\_KEYS macro to 1.

## Application description

The purpose of this demo application is to show you the usage of the FlexCAN module configured to use Flexible Data Rate and the CSEc module from the S32K142 CPU using the S32 SDK API.

- In the first part, the application will setup the board clocks, pins and other system functions.
- Then it will configure the FlexCAN module features such as FD, Btrate and Message buffers
- The application will wait for frames to be received on the configured message buffer or for an event raised by pressing one of the two buttons which will trigger a frame send to the recipient.
- Pressing SW3 button of board 1 shall trigger a CAN transfer that results in toggling the RED led on board 2.
- Pressing SW2 button of board 1 shall trigger a CAN transfer that results in toggling the GREEN led on board 2.

- Pressing both SW3 and SW2 buttons shall enable the encrypted communication. This event is signaled by the BLUE led being ON.
- The frames are sent in plain text by default.
- This demo application requires two boards, one configured as master and the other one configured as slave (see MASTER/SLAVE defines in application code).

#### Prerequisites

To run the example you will need to have the following items:

- 2 S32K142 boards
- 1 Power Adapter 12V
- 3 Dupont female to female cables
- 1 Personal Computer
- 1 PEmicro Debugger / 1 Jlink Lite Debugger (optional, users can use Open SDA)

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

#### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
CAN HIGH (*)	CAN HIGH - J13.1	CAN HIGH - J60.2
CAN LOW (*)	CAN LOW - J13.2	CAN LOW - J60.3
GND (GND)	GND - J13.4	GND - J6
BUTTON 1 (PTC13)	SW3 - wired on the board	BTN2 - wired on the board
BUTTON 2 (PTC12)	SW2 - wired on the board	BTN3 - wired on the board
RED_LED (PTD15)	RGB_RED - wired on the board	LED3 - wired on the board
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	LED2 - wired on the board
BLUE_LED (PTD0)	RGB_GREEN - wired on the board	LED1 - wired on the board

(\*) Those lines must be modulated using a transceiver, if it is not specified the boards already include the CAN transceiver. The CAN transceiver should be in Forced Normal Mode operation (default mode). To reset the CAN transceiver to default mode connect the SBC transceiver in next configuration with the board S32K142EVB-Q100 power off : • pin RSTN from SBC is held LOW • CANH(J13.1) is pulled up to VBAT(J11.2) • CANL(J13.2) is pulled down to GND(J13.4) Power on the board with external supply 12V (J16)

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **flexcan\_encrypted\_↔s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace.

## 2. Generating the S32CT configuration

First go to **Project Explorer** View in S32 DS and select the current project(**flexcan\_encrypted\_s32k142**). Then go to **Project** and click on **Generate Processor Expert Code**. Wait for the code generation to be completed before continuing to the next step.

## 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

## 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
flexcan_encrypted_s32k142_debug_flash_↔ pemicro	Debug the FLASH configuration using PEMicro debuggers
flexcan_encrypted_s32k142_debug_ram_pemicro	Debug the RAM configuration using PEMicro debuggers
flexcan_encrypted_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger J-Link debuggers
flexcan_encrypted_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger J-Link debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

## 13.3.5 CSEC BOOT PROTECTION

Basic application that presents the boot protection functionality of the CSEc module

### Note

**This example works only for CSEc enabled parts.** SIM\_SDID indicates whether CSEc is available on your device.

**The first time when running the example on the board, or after a key erase, this example should be ran from RAM.**

**After the user key was loaded using this example, any further full erase of the Flash requires a Challenge-Authentication process.** This can be done by setting the FLASH\_MODIFY macro to 1.

**After partitioning Flash for CSEc operation, using the JLink Flash configuration of any other project will not work anymore.** Workaround:

- Run csec\_boot\_protection example with FLASH\_MODIFY 1, using PEMicro Flash debug configuration

### Application description

The purpose of this demo application is to show the user how to use the boot protection feature of the Cryptographic Services Engine module from the S32K142 MCU with the S32 SDK API.

The implementation demonstrates the following:

- the enablement of the CSEc module, by showing how the Flash should be partitioned (using the Flash driver);



- configuring the MASTER\_ECU key;
- configuring the first user key, using the MASTER\_ECU key as an authorization with boot protection enabled;
- configuring and enabling secure boot;
- availability of the user key after a secure boot when the flash was modified or not.

Erasing all the configured keys (including the MASTER\_ECU key) and disabling the secure boot can be done by changing the value of the FLASH\_MODIFY macro to 1. This will place the part back into factory status (the partition command will need to be run again).

Here is a table describing the outcome based on the value of FLASH\_MODIFY and if the FLASH\_TARGET is defined:

FLASH_MODIFY	FLASH_TARGET	Result
0	UNDEFINED	Write initial code to flash
1	UNDEFINED	Write modified code to flash
0	DEFINED	Write keys and enable secure boot
1	DEFINED	Erase keys and partition flash

#### Application usage

1. The first step is to run the application from RAM having the FLASH\_MODIFY macro set to 1 in order to partition the flash. After this step, comment the #define INIT\_PHASE line
2. Load the test program to it by setting the FLASH\_MODIFY to 0 and running the application from FLASH.
3. Run the application from RAM with the FLASH\_MODIFY macro set to 0 in order to load the keys necessary for secure boot and the test key with boot protection enabled.
4. Run the application from FLASH with the FLASH\_MODIFY macro set to 0 in order to test secure boot. The secure boot process and the encryption using the test key should be successful.
5. Run the application from FLASH with the FLASH\_MODIFY macro set to 1 this time in order to modify the flash. This will result in the secure boot to fail and the test key will be unavailable so the encryption operation will be unsuccessful.
6. Run the application from FLASH with the FLASH\_MODIFY macro set to 0 in order to successfully secure boot. The test key is available again and the encryption operation is successful.
7. Set the FLASH\_MODIFY macro to 1 and run the application from RAM in order to erase the keys and flash.

#### Note

If the FLASH\_MODIFY is set to 1 at step 2 then the secure boots after the step 3 will be successful only if the FLASH\_MODIFY macro is set to 1 and unsuccessful if it is set to 0.

If an assert fails at step 3, start over at step 1, uncommenting the #define INIT\_PHASE line and setting FLASH\_MODIFY to 1.

### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 PEmicro/Jlink Lite Debugger (optional, users can use Open SDA)

### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
RED_LED (PTD15)	RGB_RED - wired on the board	J12.17 - J11.31
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	J12.16 - J11.30

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From** and select **csec\_boot\_protection\_↔s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace. Wait for the S32 Configuration was initialized and ready.

#### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

#### 3. Building the project

Select the configuration to be built **RAM** (Debug\_RAM) or **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

#### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
csec_boot_protection_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
csec_boot_protection_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
csec_boot_protection_s32k142_debug_ram_pemicro	Debug the RAM configuration using PEMicro debuggers
csec_boot_protection_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### 13.3.6 FreeMASTER

Example application showing FreeMASTER Serial Communication usage

#### Application description

The purpose of this demo application is to show you how to use the FreeMASTER serial communication using S32K142 on OpenSDA with this SDK.

This demo uses the FreeMASTER Run-Time Debugging Tool to visualise ADC conversions and allows the user to monitor the ADC sampling rate for different ADC configurations (ADC sampling time and resolution can be controlled through FreeMASTER Application Commands).

The ADC is configured to perform continuous conversions and generate an interrupt after each conversion. The LPTMR is configured to generate a periodic interrupt at 10 ms which reads the number of ADC conversions.

#### Note

This example will work only in FLASH target.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V
- 1 Dupont male to male cable
- 1 Personal Computer
- Debug probe (JLink, PEMicro, OpenSDA)
- FreeMASTER host application

#### Boards supported

The following boards are supported by this application:

- S32K142EVb-Q100
- S32K142-MB

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVb-Q100	S32K142-MB
LPUART1 TX (PTC7)	UART_TX - wired on the board	J11.26 - J20.2
LPUART1 RX (PTC6)	UART_RX - wired on the board	J11.25 - J20.5
ADC0 Input 12 (PTC14)	POT - wired on the board	J21.1 - J11.18

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **freemaster\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace. Wait for the S32 Configuration was initialized and ready.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be two debug configurations for this project:

Configuration Name	Description
freemaster_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
freemaster_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

Open the FreeMASTER project (**freemaster\_s32k142.pmp**) and set the communication parameters:

- Go to **Project -> Options -> Comm**, choose **Direct RS232** and set the COM port and speed 9600.
- Go to **Project -> Options -> MAP Files** and make sure the \*.elf file of your project's current Debug Configuration is selected and set file format to ELF/DWARF.

## Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

## Notes

FreeMASTER host application can be downloaded from NXP's website. FreeMASTER Serial Communication is included into the project (V2.0).

### 13.3.7 FreeRTOS

Demo application showing the integration of FreeRTOS and S32 SDK

#### Application description

The purpose of this demo application is to show you how to use the FreeRTOS with the S32 SDK for the S32K142 MCU.

This project defines a very simple demo that creates two tasks, one queue, and one timer. It also demonstrates how Cortex-M4 interrupts can interact with FreeRTOS tasks/timers.

This simple demo project runs 'stand alone' (without the rest of the tower system) on the Freedom Board or Validation Board, which is populated with a S32K142 Cortex-M4 microcontroller.

The idle hook function: The idle hook function demonstrates how to query the amount of FreeRTOS heap space that is remaining (see `vApplicationIdleHook()` defined in this file).

The main() Function: `main()` creates one software timer, one queue, and two tasks. It then starts the scheduler.

The Queue Send Task: The queue send task is implemented by the `prvQueueSendTask()` function in this file. `prvQueueSendTask()` sits in a loop that causes it to repeatedly block for 200 milliseconds, before sending the value 100 to the queue that was created within `main()`. Once the value is sent, the task loops back around to block for another 200 milliseconds.

The Queue Receive Task: The queue receive task is implemented by the `prvQueueReceiveTask()` function in this file. `prvQueueReceiveTask()` sits in a loop that causes it to repeatedly attempt to read data from the queue that was created within `main()`. When data is received, the task checks the value of the data, and if the value equals the expected 100, toggles the green LED. The 'block time' parameter passed to the queue receive function specifies that the task should be held in the Blocked state indefinitely to wait for data to be available on the queue. The queue receive task will only leave the Blocked state when the queue send task writes to the queue. As the queue send task writes to the queue every 200 milliseconds, the queue receive task leaves the Blocked state every 200 milliseconds, and therefore toggles the green LED every 200 milliseconds.

The LED Software Timer and the Button Interrupt: The user button BTN1 is configured to generate an interrupt each time it is pressed. The interrupt service routine switches the red LED on, and resets the LED software timer. The LED timer has a 5000 millisecond (5 second) period, and uses a callback function that is defined to just turn the LED off again. Therefore, pressing the user button will turn the LED on, and the LED will remain on until a full five seconds pass without the button being pressed.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 2 Dupont male to male cable
- 1 Personal Computer
- 1 PEMicro Multilink Debugger

#### Boards supported

The following boards are supported by this application:

- S32K14xCVD-Q064 with S32K-MB

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
RED_LED (PTD15)	RGB_RED - wired on the board	J12.17 - J11.31
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	J12.16 - J11.30
BTN (PTC13)	BTN1 - wired on the board	BTN3 - wired on the board

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **freertos\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace. Wait for the S32 Configuration to be initialized and ready.

##### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code"

##### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

##### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
freertos_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
freertos_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
freertos_s32k142_debug_ram_pemicro	Debug the RAM configuration using PEMicro debuggers
freertos_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.3.8 ADC Low Power

Demonstrates ADC trigger scheme using TRGMUX and LPIT, switches the power mode to stop and sends data using LPUART and DMA

## Application description

The purpose of this demo application is to show you the usage of a subset of the peripherals found on the S32K142 SoC.

- The application uses LPIT to trigger ADC conversions every 100ms via TRGMUX with the CPU in sleep mode. The ADC is using Hardware Compare feature to validate an conversion only if the value is greater than half of the reference voltage, in this case **VDD/2**. This way the CPU is woken up from sleep mode only if the condition is met.
- When the conversion is complete the data is transformed into a bar graph and it is sent via LPUART using DMA memory to peripheral transfer to the host PC. This way, the CPU can be put into a low power mode to reduce the energy used.

How the example works:

- Connect to the serial port using settings found in Notes section
- The user should receive an information text on the terminal emulator
- The user must press "A" or "a" for the example to run
- The potentiometer must be rotated in order to generate valid ADC conversions
- Once a valid conversion is done then a bargraph will be printed on the terminal emulator

## Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

## Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
LPUART1 TX ( <b>PTC7</b> )	UART_TX - wired on the board	J11.26 - J20.2
LPUART1 RX ( <b>PTC6</b> )	UART_RX - wired on the board	J11.25 - J20.5
ADC0 Input 12 ( <b>PTC14</b> )	POT - wired on the board	J21.1 - J11.18

Make sure the following jumpers are set:



Jumper Name	S32K142EVB-Q100
J10	Set jumper on position 2-3

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **adc\_low\_power\_↔s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace. Wait for the S32 Configuration was initialized and ready.

##### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

##### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. No RAM configuration is available because the elf file does not fit in RAM. Wait for the build action to be completed before continuing to the next step.

##### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be two debug configurations for this project:

Configuration Name	Description
adc_low_power_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
adc_low_power_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### Notes

For this example it is necessary to open a terminal emulator and configure it with:

- 115200 baud
- One stop bit
- No parity
- No flow control

#### 13.3.9 AMMCLib

Provides an example of integration of AMMCLib and S32 SDK

## Application description

The purpose of this demo application is to show you how to integrate the S32 SDK with AMMCLib.

The application starts by sending a welcome message to the terminal with instructions regarding how to select between the two parts:

1. The first part:

- The board sends a welcome message to the console with the supported operations and how to return to the menu.
- It uses LPUART to communicate with the user and get the simple mathematical expressions.
- The received expression is then interpreted and the result is calculated using mathematical functions from AMMCLib and then sent back to the terminal as a floating point with a precision of 4.

2. The second part:

- The board sends a welcome message to the console with further instructions and how to return to the menu.
- It uses LPTMR to generate samples of a sinusoidal signal, once every 1 ms, using trigonometric functions from the AMMCLib.  
The sinusoidal signal can be seen using the FreeMASTER host application.  
Calculated signal samples are then scaled to be in the range of the FTM PWM duty cycle and are used to change the intensity of the RGB leds.  
The frequency of each sine can be controlled with the command `set_RGB_frequency()` from FreeMASTER project. The frequency sent is in mHz and the default value is 0,25Hz.
- Also, it implements an exponential moving average filter using the Potentiometer on ADC channel 12 as input.  
The output of the filter can be seen using the FreeMASTER host application.  
The filter's smoothing factor ( $\lambda$ ) can be controlled using the command `set_FilterMA_lambda()` from FreeMASTER project.

## Note

For more detailed information on the AMMCLib's functions please consult the available documentation.

## Prerequisites

To run the example, you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from USB)
- 1 Personal Computer
- Debug probe (JLink, PEmicro, OpenSDA)
- FreeMASTER host application
- UART to USB converter if it is not included on the target board. (Please consult your boards documentation to check if UART-USB converter is present).

## Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

#### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
FTM0 Channel 0 ( <b>PTD15</b> )	RGB_RED - wired on the board	J12.18 - J11.31
FTM0 Channel 1 ( <b>PTD16</b> )	RGB_GREEN - wired on the board	J12.17 - J11.32
FTM0 Channel 2 ( <b>PTD0</b> )	RGB_BLUE - wired on the board	J12.31 - J11.30
ADC0 Input 12 ( <b>PTC14</b> )	POT - wired on the board	J21.1 - J11.18

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File** -> **New** -> **S32DS Project From** and select **ammclib\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace. Wait for the S32 Configuration was initialized and ready.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be two debug configurations for this project:

Configuration Name	Description
<b>ammclib_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers
<b>ammclib_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

A terminal emulator configured with the following communication parameters is needed by this application:

- 9600 Baud rate
- 8 Data bits
- 1 Stop bit
- No parity
- No flow control

For the first part of the application follow the instructions in the terminal.

For the second part of the application you need to:

1. exit the mathematical section by typing **exit** in the terminal
2. select second section by typing **2** in the terminal

3. disconnect the terminal and start FreeMASTER.

Open the FreeMASTER project (**ammclib.pmp**) and set the communication parameters:

- Go to **Project -> Options -> Comm**, choose **Direct RS232** and set the COM port and speed 9600.
- Go to **Project -> Options -> MAP Files** and make sure the \*.elf file of your project's current Debug Configuration is selected and set file format to ELF/DWARF.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### Notes

FreeMASTER host application can be downloaded from [NXP's website](#).

FreeMASTER Serial Communication is included into the project (V2.0).

#### 13.3.10 LIN MASTER

Example that shows the usage of the LIN driver in master mode

#### Application description

This example demonstrates the LIN communication between S32K142 Master and Slave using unconditional frames.

- The Master SeatECU is in NormalTable schedule table and it uses the LIN frame Motor1State\_Cycl to receive temperature signal Motor1Temp from Slave Motor1 and send selection signal Motor1Selection to Slave Motor1 by frame Motor1Control. The first turn on GREEN\_LED, then 5s GREEN\_LED and BLUE\_LED will toggle alternately.
- If value of temperature signal is higher than MOTOR1\_OVER\_TEMP value, Master SeatECU will send STOP command through Motor1Selection signal to stop motor and turn on RED\_LED.
- If value of temperature signal is in range from MOTOR1\_MAX\_TEMP value to MOTOR1\_OVER\_TEMP value, master SeatECU will send DECREASE MOTOR SPEED command through Motor1Selection signal to reduce motor speed and turn on BLUE\_LED.
- If value of temperature signal is lower than MOTOR1\_MAX\_TEMP value, master will send INCREASE MOTOR SPEED command through Motor1Selection signal to increase motor speed and turn on GREEN\_LED.
- When users press button BUTTON 0 on the Master board, the Master SeatECU switches its schedule table to go-to-sleep table. So the Slave and Master enter sleep mode, RGB LEDS are off.
- When LIN cluster is in sleep mode, users press button BUTTON 1 on the Master board, the Master board sends a wakeup signal to wakeup slave nodes, then switches its table to NormalTable.

#### Prerequisites

To run the example you will need to have the following items:

- 2 S32K142 boards
- 1 Power Adapter 12V
- 2 Dupont female to female cable
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

## Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100-Master	S32K142EVB-Q100-Slave
BUTTON 0 (PTC12)	SW2 - wired on the board	SW2 - wired on the board
BUTTON 1 (PTC13)	SW3 - wired on the board	SW3 - wired on the board
RED_LED (PTD15)	RGB_RED - wired on the board	RGB_RED - wired on the board
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	RGB_GREEN - wired on the board
BLUE_LED (PTD0)	RGB_BLUE - wired on the board	RGB_BLUE - wired on the board
LIN (*)	J11-1 - LIN	J11-1 - LIN
RX-WORKAROUND	J5-18 - J6-9	J5-18 - J6-9
TX-WORKAROUND	J5-16 - J6-11	J5-16 - J6-11
GND (GND)	J11-4 - GND	J11-4 - GND

(\*) Those lines must be modulated using a transceiver, if it is not specified the boards already include the LIN transceiver

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **lin\_master\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**lin\_master\_s32k142**). Select the "ConfigTools" menu then click on the desired configuration (Pins, Clocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<b>lin_master_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using Segger Jlink debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

**Note**

For more detailed information related to S32 Design Studio usage please consult the available documentation.

**Notes**

The default configuration of this example supports S32K142 EVB RevD.

RX-WORKAROUND and TX-WORKAROUND Pin Function do not need to be set up when using RevD with the Default Configuration of example.

**13.3.11 LIN SLAVE**

Example that shows the usage of the LIN driver in slave mode

**Application description**

This example demonstrates the LIN communication between S32K142 Master and Slave using unconditional frames.

- The Master SeatECU is in NormalTable schedule table and it uses the LIN frame Motor1State\_Cycl to receive temperature signal Motor1Temp from Slave Motor1 and send selection signal Motor1Selection to Slave Motor1 by frame Motor1Control. The first turn on GREEN\_LED, then 5s GREEN\_LED and BLUE\_LED will toggle alternately.
- When user press button BUTTON 0 on the Slave board, value of temperature signal (Motor1\_temp) will be increased 60 unit.
- When user press button BUTTON 1 on the Slave board, value of temperature signal will be set to value which is lower MOTOR1\_MAX\_TEMP value and turn on GREEN\_LED.
- If value of temperature signal is higher than MOTOR1\_OVER\_TEMP value, Master SeatECU will send STOP command through Motor1Selection signal to stop motor and turn on RED\_LED.
- If value of temperature signal is in range from MOTOR1\_MAX\_TEMP value to MOTOR1\_OVER\_TEMP value, master SeatECU will send DECREASE MOTOR SPEED command through Motor1Selection signal to reduce motor speed and turn on BLUE\_LED.
- If value of temperature signal is lower than MOTOR1\_MAX\_TEMP value, master will send INCREASE MOTOR SPEED command through Motor1Selection signal to increase motor speed and turn on GREEN\_LED.
- When users press button BUTTON 0 on the Master board, the Master SeatECU switches its schedule table to go-to-sleep table. So the Slave and Master enter sleep mode, all LEDs are off.
- When LIN cluster is in sleep mode, users press button BUTTON 1 on the Master board, the Master board sends a wakeup signal to wakeup slave nodes, then switches its table to NormalTable.

**Prerequisites**

To run the example you will need to have the following items:

- 2 S32K142 boards
- 1 Power Adapter 12V
- 2 Dupont female to female cable
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

## Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100-Slave	S32K142EVB-Q100-Master
BUTTON 0 (PTC12)	SW2 - wired on the board	SW2 - wired on the board
BUTTON 1 (PTC13)	SW3 - wired on the board	SW3 - wired on the board
RED_LED (PTD15)	RGB_RED - wired on the board	RGB_RED - wired on the board
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	RGB_GREEN - wired on the board
BLUE_LED (PTD0)	RGB_BLUE - wired on the board	RGB_BLUE - wired on the board
LIN (*)	J11-1 - LIN	J11-1 - LIN
RX-WORKAROUND	J5-18 - J6-9	J5-18 - J6-9
TX-WORKAROUND	J5-16 - J6-11	J5-16 - J6-11
GND (GND)	J11-4 - GND	J11-4 - GND

(\*) Those lines must be modulated using a transceiver, if it is not specified the boards already include the LIN transceiver

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **lin\_slave\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**lin\_slave\_s32k142**). Select the "ConfigTools" menu then click on the desired configuration (Pins, Clocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<b>lin_slave_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.



**Note**

For more detailed information related to S32 Design Studio usage please consult the available documentation.

**Notes**

The default configuration of this example supports S32K142 EVB RevD.

If you want to use S32K-MB please remove EVB macro.

RX-WORKAROUND and TX-WORKAROUND Pin Function do not need to be set up when using RevD with the Default Configuration of example.

**13.3.12 Structural Core Self Test Example**

Basic application that presents the project scenarios for S32 SDK

**Application description**

The purpose of this demo application is to show you how to integrate the S32 SDK with sCST.

- The application will run the core self tests from the Structural Core Self Test library and will report the result using the user leds.
- Please consult the sCST manual for more information about the library.

**Note**

This application uses a modified version of the linker file which defines the section used by the library. As a consequence, the application will only run in flash.

**Prerequisites**

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- Debug probe (JLink, PEmicro, OpenSDA)

**Boards supported**

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

**Hardware Wiring**

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
RED_LED (PTD15)	RGB_RED - wired on the board	J12.17 - J11.31

GREEN_LED (PTD16)	RGB_GREEN - wired on the board	J12.16 - J11.30
-------------------	--------------------------------	-----------------

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File** -> **New S32DS Project From...** and select **scst\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace. Wait for the S32 Configuration to be initialized and ready.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has a blue chip symbol on the top of the toolbar). In the S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be two debug configurations for this project:

Configuration Name	Description
scst_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
scst_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEmicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

## Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

## 13.4 Driver Examples

Applications that show the user how to initialize the peripherals for the basic use cases

There are currently examples for the following categories:

*Click on one of the categories to see the available projects*

- [Analog Driver Examples](#)
- [Communication Driver Examples](#)
- [System Driver Examples](#)
- [Timer Driver Examples](#)

#### 13.4.1 Analog Driver Examples

Applications that show the user how to initialize the analog peripherals

There are currently driver examples with the following modules:

*Click on one of the module to see the available projects*

- [ADC Hardware Trigger](#)
- [ADC PAL example](#)
- [ADC Software Trigger](#)
- [CMP DAC](#)

#### 13.4.2 ADC Hardware Trigger

How to trigger the ADC by hardware

##### Application description

The purpose of this demo application is to show you the usage of the ADC module triggered in hardware by the Programmable Delay Block from the S32K142 CPU using the S32 SDK API.

- The application uses PDB to trigger ADC conversions every 1s.
- When the conversion is complete the data is sent to the host PC using LPUART.

##### See also

PDB\_Example\_group

For alternate ADC Hardware triggering scheme see ADC\_LOW\_POWER\_group

##### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)
- UART to USB converter if it is not included on the target board. (Please consult your boards documentation to check if UART-USB converter is present).

##### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
LPUART1 TX ( <b>PTC7</b> )	UART_TX - wired on the board	J11.26 - J20.2
LPUART1 RX ( <b>PTC6</b> )	UART_RX - wired on the board	J11.25 - J20.5
ADC0 Input 12 ( <b>PTC14</b> )	POT - wired on the board	J21.1 - J11.18

Make sure the following jumpers are set:

Jumper Name	S32K142EVB-Q100
J10	Set jumper on position 2-3

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **adc\_hwtrigger\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace. Wait for the S32 Configuration to be initialized and ready.

##### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

##### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

##### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
adc_hwtrigger_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
adc_hwtrigger_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
adc_hwtrigger_s32k142_debug_ram_pemicro	Debug the RAM configuration using PEmicro debuggers
adc_hwtrigger_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEmicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### Notes

For this example it is necessary to open a terminal emulator and configure it with:

- 115200 baud

- One stop bit
- No parity
- No flow control

### 13.4.3 ADC PAL example

Example for ADC PAL usage

#### Application description

The purpose of this demo application is to present the basic functionality of the Analog to Digital Converter Peripheral Abstraction Layer (ADC PAL) on S32K14x MCU.

The application uses ADC PAL to trigger multiple executions of two groups of ADC conversions: first group configured for SW triggering and second group for HW triggering. For each execution of a group of conversions, an average conversion value is computed in SW, and the average value is printed on UART.

example is divided in 2 parts:

- Part 1: SW triggered group of conversions  
After each complete execution of the group, results are read, the average value is calculated and printed to console. A delay is inserted and then the SW group is triggered again. The process is repeated for a fixed number of iterations.
- Part 2: HW triggered group of conversions  
LPTMR is configured to provide a trigger event with a fixed periodicity. The selected HW group is enabled. After each complete execution of the group, results are read, the average value is calculated and printed to console. After a fixed number of iterations, the HW trigger group of conversions is disabled, and the LPTMR is stopped.

Note: both HW and SW triggered groups are configured to run all conversions on a single ADC InputChannel, because the development board contains only a single potentiometer connected to the MCU. However, the ADC PAL supports different InputChannels to be used in the same group. For more details please refer to the ADC PAL documentation.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)
- UART to USB converter if it is not included on the target board. (Please consult your boards documentation to check if UART-USB converter is present).

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

#### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
LPUART1 TX (PTC7)	UART_TX - wired on the board	J11.26 - J20.2
LPUART1 RX (PTC6)	UART_RX - wired on the board	J11.25 - J20.5
ADC0 Input 12 (PTC14)	POT - wired on the board	J21.1 - J11.18

Make sure the following jumpers are set:

Jumper Name	S32K142EVB-Q100
J10	Set jumper on position 2-3

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **adc\_pal\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace. Wait for the S32 Configuration to be initialized and ready.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be two debug configurations for this project:

Configuration Name	Description
adc_pal_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
adc_pal_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEmicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

## Note

The Debug\_RAM configuration could not be accommodated for the current example, due to the small RAM footprint present on S32K142.

For more detailed information related to S32 Design Studio usage please consult the available documentation.

## Notes

For this example it is necessary to open a terminal emulator and configure it with:

- 115200 baud
- One stop bit



- No parity
- No flow control

#### 13.4.4 ADC Software Trigger

How to trigger ADC by software

##### Application description

The purpose of this demo application is to show you the usage of the ADC module triggered by software from the S32K142 CPU using the S32 SDK API.

- The application uses software to trigger ADC conversions every 1s.
- When the conversion is complete the data is sent to the host PC using LPUART.

##### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)
- UART to USB converter if it is not included on the target board. (Please consult your boards documentation to check if UART-USB converter is present).

##### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

##### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
LPUART1 TX ( <b>PTC7</b> )	UART_TX - wired on the board	J11.26 - J20.2
LPUART1 RX ( <b>PTC6</b> )	UART_RX - wired on the board	J11.25 - J20.5
ADC0 Input 12 ( <b>PTC14</b> )	POT - wired on the board	J21.1 - J11.18

Make sure the following jumpers are set:

Jumper Name	S32K142EVB-Q100
J10	Set jumper on position 2-3

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **adc\_swtrigger\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace. Wait for the S32 Configuration to be initialized and ready.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<b>adc_swtrigger_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Segger Jlink debuggers
<b>adc_swtrigger_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers
<b>adc_swtrigger_s32k142_debug_ram_pemicro</b>	Debug the RAM configuration using PEMicro debuggers
<b>adc_swtrigger_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### Notes

For this example it is necessary to open a terminal emulator and configure it with:

- 115200 baud
- One stop bit
- No parity
- No flow control

### 13.4.5 CMP DAC

Driver examples showing the basic usage scenario of the CMP

### Application description

The purpose of this demo application is to show you how to use the Analog Comparator of the S32K142 MCU using the S32 SDK API.

The Comparator is configured to compare analog input 0(AIN0) with half the reference voltage generated with the internal DAC. Based on the input from the potentiometer the LEDs light by the following rules:

- 1)  $V_{in} < \text{DAC voltage}$  : RED on, GREEN off
- 2)  $V_{in} > \text{DAC voltage}$  : RED off, GREEN on
- 3) Unknown state : RED on, GREEN on

### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
RED_LED (PTD15)	RGB_RED - wired on the board	J5.1 - J11.31
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	J5.3 - J11.32
CMP Input 0 (PTA0)	J4.14 - J5.7	J21.1 - J9.31

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **cmp\_dac\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

#### 2. Generating the S32 Configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated.

The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar).

In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components.

Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user.

Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
cmp_dac_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
cmp_dac_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
cmp_dac_s32k142_debug_ram_pemicro	Debug the RAM configuration using PEMicro debuggers
cmp_dac_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.6 Communication Driver Examples

Applications that show the user how to initialize the communication peripherals

There are currently driver examples with the following modules:

*Click on one of the module to see the available projects*

- [FLEXIO SPI](#)
- [LPUART Echo](#)
- [UART PAL ECHO](#)
- [CAN PAL](#)
- [LPSPI Transfer](#)
- [LPSPI DMA](#)
- [SPI PAL](#)
- [I2C PAL](#)
- [I2S PAL MASTER](#)

- [I2S PAL SLAVE](#)
- [FLEXIO I2C](#)
- [FLEXIO I2S MASTER](#)
- [FLEXIO I2S SLAVE](#)
- [FLEXIO UART](#)
- [LPI2C MASTER](#)
- [LPI2C SLAVE](#)
- [LIN MASTER BAREMETAL](#)
- [LIN SLAVE BAREMETAL](#)
- [SBC UJA116x](#)

#### 13.4.7 FLEXIO SPI

Example application showing FlexIO SPI driver usage

##### Application description

The purpose of this demo application is to show you the usage of the FlexIO SPI driver found on the S32K144 SoC using S32 SDK API.

The application uses FlexIO SPI driver to make a data transfer of a defined size. The slave device for this example is a second FlexIO SPI driver using the same FlexIO instance, which is configured to act as a bus slave. The slave and master buffers will be checked after each transfer by the application, user shall check **isTransferOk** variable to see if the transmissions are successful(Green led will turn on), otherwise red led will turn on.

##### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board can't be powered from the USB)
- 4 Dupont male to male cable
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

## Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100
FLEXIO_MASTER CS ( <b>PTA1</b> )	J5.5 - J6.19
FLEXIO_MASTER SCK ( <b>PTA0</b> )	J5.7 - J6.17
FLEXIO_MASTER MOSI ( <b>PTD0</b> )	J2.6 - J1.1
FLEXIO_MASTER MISO ( <b>PTA11</b> )	J1.2 - J1.3
FLEXIO_SLAVE SS ( <b>PTA9</b> )	J6.19 - J5.5
FLEXIO_SLAVE SCK ( <b>PTA8</b> )	J6.17 - J5.7
FLEXIO_SLAVE MOSI ( <b>PTA2</b> )	J1.1 - J2.6
FLEXIO_SLAVE MISO ( <b>PTA3</b> )	J1.3 - J1.2
RED_LED ( <b>PTD15</b> )	RGB_RED - wired on board
GREEN_LED ( <b>PTD16</b> )	RGB_GREEN - wired on board

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **flexio\_spi\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There are four debug configurations for this project:

Configuration Name	Description
<b>flexio_spi_s32k142_ram_pemicro</b>	Debug the RAM configuration using PEMicro debuggers
<b>flexio_spi_s32k142_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers

<b>flexio_spi_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Jlink debuggers
<b>flexio_spi_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Jlink debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.8 LPUART Echo

Example application using the LPUART driver

#### Application description

The purpose of this demo application is to show you how to use the Low Power UART from the S32K142 CPU using the S32 SDK API.

- The welcome message is sent via UART: "This example is an simple echo using LPUART it will send back any character you send to it. The board will greet you if you send 'Hello Board' Now you can begin typing:" - User shall send "Hello Board" string. If the board receives the user's string, then the "Hello World" string shall be sent again. User need to add EOL character to string which will be sent to board.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 PEmicro debugger
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

#### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
LPUART1 TX (PTC7)	UART_TX - wired on the board	J11.26 - J20.2

LPUART1 RX (PTC6)	UART_RX - wired on the board	J11.25 - J20.5
-------------------	------------------------------	----------------

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **lpuart\_echo\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has a blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
lpuart_echo_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
lpuart_echo_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
lpuart_echo_s32k142_debug_ram_pemicro	Debug the RAM configuration using PEMicro debuggers
lpuart_echo_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

## Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

## Notes

For this example it is necessary to open a terminal emulator and configure it with:

- 9600 baud
- One stop bit
- No parity
- No flow control
- '\n' line ending



### 13.4.9 UART PAL ECHO

Basic application that presents the project scenarios for S32 SDK

#### Application description

The purpose of this demo is to show the user how UART PAL works over FLEXIO\_UART or LPUART peripherals. The user can choose whether to use FLEXIO\_UART or LPUART (see USE\_FLEXIO\_UART define from The board sends a welcome message to the console with further instructions.)

- The welcome message is sent via UART: "This example is an simple echo using uart\_pal\_echo it will send back any character you send to it. The board will greet you if you send 'Hello!' Now you can begin typing:" - User shall send "Hello!" string. If the board receives the user's string, then the "Hello World!" string shall be sent again. User need to add EOL character to string which will be sent to board. Green led(devkit) or led 1(Motherboard) shall be turned on if the communication is done over FLEXIO\_UART; similarly the led shall be turned off if the communication is done over LPUART.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 2 Dupont male to male cable
- 1 Personal Computer
- 1 PEmicro/Jlink Lite Debugger (optional, users can use Open SDA)

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

#### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	
LED1 (PTC1)		JP50 - jump 50 on motherboard
LPUART1 TX (PTC7)	UART_TX - wired on the board	J11.26 - J20.2
LPUART1 RX (PTC6)	UART_RX - wired on the board	J11.25 - J20.5
FLEXIO_UART RX (PTA11)	J1.2 - J4.4	J9.22 - J20.5
FLEXIO_UART TX (PTA0)	J5.7 - J4.2	J9.31 - J20.2

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **hello\_world\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace. Wait for the S32 Configuration to be initialized and ready.

## 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has a blue chip symbol on the top of the toolbar). In the S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

## 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

## 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
uart_pal_echo_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
uart_pal_echo_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEmicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### Notes

For this example it is necessary to open a terminal emulator and configure it with:

- 115200 baud
- One stop bit
- No parity
- No flow control
- '\n' line ending

## 13.4.10 CAN PAL

Demo application showing the CAN PAL functionalities

### Application description

The purpose of this demo application is to show you the usage of the CAN PAL module configured to use Flexible Data Rate from the S32K142 CPU using the S32 SDK API.

- In the first part, the application will setup the board clocks, pins and other system functions such as SBC if the board uses this module as a CAN transceiver.

- Then it will configure the CAN PAL module features such as FD, Btrrate and buffers
- The application will wait for frames to be received on the configured buffer or for an event raised by pressing one of the two buttons which will trigger a frame send to the recipient.
- Pressing SW3 button of board 1 shall trigger a CAN transfer that results in toggling the RED led\ on board 2.
- Pressing SW2 button of board 1 shall trigger a CAN transfer that results in toggling the GREEN led on board 2.
- This demo application requires two boards, one configured as master and the other one configured as slave (see MASTER/SLAVE defines in application code).

#### Prerequisites

To run the example you will need to have the following items:

- 2 S32K142 boards
- 1 Power Adapter 12V
- 3 Dupont female to female cables
- 1 Personal Computer
- 1 Jlink Lite Debugger/ PEMicro Debugger (optional, users can use Open SDA for S32K142EVB-Q100)

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

#### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
CAN HIGH (*)	CAN HIGH - J13.1	CAN HIGH - J60.2
CAN LOW (*)	CAN LOW - J13.2	CAN LOW - J60.3
GND (GND)	GND - J13.4	GND - J6
BUTTON 1 (PTC13)	SW3 - wired on the board	BTN2 - wired on the board
BUTTON 2 (PTC12)	SW2 - wired on the board	BTN3 - wired on the board
RED_LED (PTD15)	RGB_RED - wired on the board	LED3 - wired on the board
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	LED2 - wired on the board

(\*) Those lines must be modulated using a transceiver, if it is not specified the boards already include the CAN transceiver.

The CAN transceiver should be in Forced Normal Mode operation (default mode).

To reset the CAN transceiver to default mode connect the SBC transceiver in next configuration with the board S32K142EVB-Q100 power off:

- pin RSTN from SBC is held LOW
- CANH(J13.1) is pulled up to VBAT(J11.2)
- CANL(J13.2) is pulled down to GND(J13.4)

Power on the board with external supply 12V (J16)

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **can\_pal\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace. Wait for the S32 Configuration to be initialized and ready.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has a blue chip symbol on the top of the toolbar). In the S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **RAM** (Debug\_RAM) or **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
can_pal_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
can_pal_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
can_pal_s32k142_debug_ram_pemicro	Debug the RAM configuration using PEmicro debuggers
can_pal_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEmicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.11 LPSPI Transfer

Driver example that will show the LPSPI Master and Slave functionalities

#### Application description

The purpose of this application is to show the user how to use the Low Power Serial Peripheral Interface on the S32K142 using the S32 SDK API.

- The application uses two on-board instances of LPSPI, one in master configuration and the other one is slave to communicate data via the SPI bus. Data will be gathered periodically from the ADC input and will be sent to the master device which transforms it into a PWM signal. In this way the potentiometer controls the LED intensity.

### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 4(6) Dupont male to male cables
- 1 Personal Computer
- 1 Jlink Lite Debugger or 1 PEmicro Debugger (optional, users can use Open SDA)

### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
LPSP10 CS ( <b>PTB0</b> )	J4.5 - J1.14	J10.31 - J12.30
LPSP10 SCK ( <b>PTB2</b> )	J2.11 - J4.8	J10.29 - J12.31
LPSP10 MOSI ( <b>PTE1</b> )	J5.14 - J6.3	J13.32 - J12.29
LPSP10 MISO ( <b>PTB4</b> )	J2.7 - J6.2	J10.27 - J12.32
LPSP11 CS ( <b>PTD3</b> )	J4.5 - J1.14	J10.31 - J12.30
LPSP11 SCK ( <b>PTB14</b> )	J2.11 - J4.8	J10.29 - J12.31
LPSP11 MOSI ( <b>PTD2</b> )	J5.14 - J6.3	J13.32 - J12.29
LPSP11 MISO ( <b>PTD1</b> )	J2.7 - J6.2	J10.27 - J12.32
ADC0 Input 12 ( <b>PTC14</b> )	POT - wired on the board	J21.1 - J11.18
LED ( <b>PTD15</b> )	LED - wired on the board	J12.18 - J5.13

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **lpspi\_transfer\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

#### 2. Generating the S32 Configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated.

The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar).

In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components.

Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user.

Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be two debug configurations for this project:

Configuration Name	Description
lpspi_transfer_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
lpspi_transfer_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### Notes

With 12V power supply, it is recommended to provide 3.3V to MCU. Jump J10.1 - J10.2 should be connected.

### 13.4.12 LPSPI DMA

Driver example that will show the LPSPI Master and Slave functionalities

#### Application description

The purpose of this application is to show you how to use the Low Power Serial Peripheral Interface on the S32K142 using the S32 SDK API.

The application uses two on board instances of LPSPI, one in master configuration and the other one is slave to communicate data via the SPI bus using DMA.

To check if the transmission is successful the user has to verify that the data sent is the same as the received data. If transfer is successful, RED led will be on, otherwise it will be off.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 4 Dupont male to male cables
- 1 Personal Computer
- 1 Jlink Lite Debugger or 1 PEMicro Debugger (optional, users can use Open SDA)

### Boards supported

The following boards are supported by this application:

- S32K142EVb-Q100
- S32K142-MB

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVb-Q100	S32K142-MB
LPSPi_MASTER CS ( <b>PTB0</b> )	J4.5 - J6.4	PTB0 - PTD3
LPSPi_MASTER SCK ( <b>PTB2</b> )	J2.11 - J2.6	PTB2 - PTD0
LPSPi_MASTER MOSI ( <b>PTB4</b> )	J2.7 - J6.2	PTB4 - PTD1
LPSPi_MASTER MISO ( <b>PTE1</b> )	J5.14 - J6.3	PTE1 - PTD2
LPSPi_SLAVE SS ( <b>PTD3</b> )	J6.4 - J4.5	PTD3 - PTB0
LPSPi_SLAVE SCK ( <b>PTD0</b> )	J2.6 - J2.11	PTD0 - PTB2
LPSPi_SLAVE MOSI ( <b>PTD1</b> )	J6.2 - J2.7	PTD1 - PTB4
LPSPi_SLAVE MISO ( <b>PTD2</b> )	J6.3 - J5.14	PTD2 - PTE1
LED ( <b>PTD15</b> )	LED - wired on the board	J12.18 - J5.1

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **lpspi\_dma\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

#### 2. Generating the S32 Configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated.

The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar).

In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components.

Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user.

Make the desired changes (if any) then click "Update Code".

#### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

#### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<b>lpspi_dma_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Segger Jlink debuggers

<b>lpspi_dma_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers
<b>lpspi_dma_s32k142_debug_ram_pemicro</b>	Debug the RAM configuration using PEMicro debuggers
<b>lpspi_dma_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### 13.4.13 SPI PAL

Driver example using SPI

#### Application description

The purpose of this application is to show you how to use the LPSPI and FLEXIO Interfaces on the S32K142 using the S32 SDK API.

The application uses one board instance of LPSPI in slave configuration and one board instance of FLEXIO in master configuration to communicate data via the SPI bus using interrupts. It also verifies that the data sent is the same as the received data. If transfer is successful, RED led will be on, otherwise it will be off.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board can't be powered from the USB)
- 1 Personal Computer
- 4 Dupont male to male cable
- 1 Jlink Lite Debugger or 1 PEMicro Debugger (optional, users can use Open SDA)

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

#### Hardware Wiring

The following connections must be done to for this example application to work:



PIN FUNCTION	S32K142EVb-Q100	S32K142-MB
FLEXIO_MASTER CS ( <b>PTA1</b> )	J5.5 - J4.5	PTA1 - PTB0
FLEXIO_MASTER SCK ( <b>PTA0</b> )	J5.7 - J2.11	PTA0 - PTB2
FLEXIO_MASTER MOSI ( <b>PTD1</b> )	J6.2 - J2.9	PTD1 - PTB3
FLEXIO_MASTER MISO ( <b>PTA2</b> )	J1.1 - J4.7	PTA2 - PTB1
LPSPI_SLAVE SS ( <b>PTB0</b> )	J5.5 - J4.5	PTB0 - PTA1
LPSPI_SLAVE SCK ( <b>PTB2</b> )	J5.7 - J2.11	PTB2 - PTA0
LPSPI_SLAVE MOSI ( <b>PTB3</b> )	J6.2 - J2.9	PTB3 - PTD1
LPSPI_SLAVE MISO ( <b>PTB1</b> )	J1.1 - J4.7	PTB1 - PTA2
LED ( <b>PTD15</b> )	LED - wired on the board	J12.18 - J5.1

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **spi\_pal\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace.

##### 2. Generating the S32 Configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated.

The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar).

In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components.

Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user.

Make the desired changes (if any) then click "Update Code".

##### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

##### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be two debug configurations for this project:

Configuration Name	Description
<b>spi_pal_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers
<b>spi_pal_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEmicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.14 I2C PAL

Driver example using I2C

## Application description

The purpose of this application is to show you how to use the LPI2C and FLEXIO Interfaces on the S32K142 using the S32 SDK API.

The application uses one board instance of LPI2C in slave configuration and one board instance of FLEXIO in master configuration to communicate data via the I2C bus using interrupts.

The RED or GREEN led will be turn on or turn off depending on the check result. Red led will turn on if data does not match. Green led will turn on if then data is transfered correctly.

## Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board can't be powered from the USB)
- 1 Personal Computer
- 4 Dupont male to male cable
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

## Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
FLEXIO SDA ( <b>PTE15</b> )	J1.1 - J5.4	J9.29 - J12.31
FLEXIO SCL ( <b>PTA11</b> )	J1.3 - J1.2	J9.30 - J9.22
LPI2C SDA ( <b>PTA2</b> )	J1.1 - J5.4	J9.29 - J12.31
LPI2C SCL ( <b>PTA3</b> )	J1.3 - J1.2	J9.30 - J9.22

The pull-up resistors should be connected one between VCC(J3.7) and SDA pin(J1.1) and the second one between VCC(J3.7) and SCL pin(J1.3).

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **i2c\_pal\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace. Wait for the S32 Configuration was initialized and ready.

## 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

## 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

## 4. Running the project

Go to **Run** and select **Debug Configurations**. There are two debug configurations for this project:

Configuration Name	Description
i2c_pal_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers
i2c_pal_s32k142_debug_flash_jlink	Debug the FLASH configuration using Jlink debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### 13.4.15 I2S PAL MASTER

Driver example using I2S

#### Application description

The purpose of this application is to show you how to use the i2s\_pal driver on the S32K142.

The application uses one instance of FLEXIO in slave board and one instance of FLEXIO in master board to communicate data via the I2S bus using both of interrupts and DMA The application will work in conjunction with the i2s\_pal\_slave demo on S32K1xx.

The application displays on the host PC terminal a menu in which the user can select to: For Slave board: "Press: 1) [Slave] Send data 2) [Slave] Received data Enter your input:"

For Master board: "Press: 1) [Master] Send data 2) [Master] Received data Enter your input:"

Select Send/Receive on Slave first. After that select Receive/Send on Master.

The master buffers and slave buffers will be checked after each transfer by the application, RED or GREEN led will be lit depend on the check result. Red led will turn on if data does not match. Green led will turn on if then data is transferred correctly.

#### Prerequisites

To run the example you will need to have the following items:

- 2 S32K142 board
- 1 Personal Computer

- 4 male to male jump wires
- 2 J-link Lite Debugger (optional, users can use Open SDA)
- 2 Power Adapter 12V (if the board can't be powered from the USB)

### Boards supported

The following boards are supported by this application:

- S32K142-EVB

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142-EVB MASTER	S32K142-EVB SLAVE
FLEXIO SCK	J6.5 (PTD9)	J1.1(PTA2)
FLEXIO WS	J1.2 (PTA11)	J1.3(PTA3)
FLEXIO MASTER TX - SLAVE RX	J5.7 (PTA0)	J6.19(PTA9)
FLEXIO MASTER RX - SLAVE TX	J5.5 (PTA1)	J6.17(PTA8)
RED_LED (PTD15)	RGB_RED - wired on board	RGB_RED - wired on board
GREEN_LED (PTD16)	RGB_GREEN - wired on board	RGB_GREEN - wired on board
UART	Wired on board	Wired on board

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **i2s\_pal\_master\_s32k142**. Then click on **Finish**.  
The project should now be copied into you current workspace.

#### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**i2s\_pal\_master\_s32k142**). Select the "ConfigTools" menu then click on the desired configuration (Pins, Cocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

#### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

#### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be two debug configurations for this project:

Configuration Name	Description
i2s_pal_master_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers

<b>i2s_pal_master_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers
---	---

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### Notes

For this example it is necessary to open a terminal emulator and configure it with:

- 9600 baud
- One stop bit
- No parity
- No flow control
- '\n' line ending

#### 13.4.16 I2S PAL SLAVE

Driver example using I2S

#### Application description

The purpose of this application is to show you how to use the i2s\_pal driver on the S32K142.

The application uses one instance of FLEXIO in slave board and one instance of FLEXIO in master board to communicate data via the I2S bus using both of interrupts and DMA. The application will work in conjunction with the i2s\_pal\_master demo on S32K1xx.

The application displays on the host PC terminal a menu in which the user can select to: For Slave board: "Press: 1) [Slave] Send data 2) [Slave] Received data Enter your input:"

For Master board: "Press: 1) [Master] Send data 2) [Master] Received data Enter your input:"

Select Send/Receive on Slave first. After that select Receive/Send on Master.

The master buffers and slave buffers will be checked after each transfer by the application, RED or GREEN led will be lit depend on the check result. Red led will turn on if data does not match. Green led will turn on if then data is transferred correctly.

#### Prerequisites

To run the example you will need to have the following items:

- 2 S32K142 board
- 1 Personal Computer
- 4 male to male jump wires
- 2 J-link Lite Debugger (optional, users can use Open SDA)
- 2 Power Adapter 12V (if the board can't be powered from the USB)

## Boards supported

The following boards are supported by this application:

- S32K142-EVB

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142-EVB MASTER	S32K142-EVB SLAVE
FLEXIO SCK	J6.5 (PTD9)	J1.1(PTA2)
FLEXIO WS	J1.2 (PTA11)	J1.3(PTA3)
FLEXIO MASTER TX - SLAVE RX	J5.7 (PTA0)	J6.19(PTA9)
FLEXIO MASTER RX - SLAVE TX	J5.5 (PTA1)	J6.17(PTA8)
RED_LED (PTD15)	RGB_RED - wired on board	RGB_RED - wired on board
GREEN_LED (PTD16)	RGB_GREEN - wired on board	RGB_GREEN - wired on board
UART	Wired on board	Wired on board

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **i2s\_pal\_slave\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**i2s\_pal\_slave\_s32k142**). Select the "ConfigTools" menu then click on the desired configuration (Pins, Cocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be two debug configurations for this project:

Configuration Name	Description
i2s_pal_slave_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
i2s_pal_slave_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

## Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

## Notes

For this example it is necessary to open a terminal emulator and configure it with:

- 9600 baud
- One stop bit
- No parity
- No flow control
- "\n" line ending

### 13.4.17 FLEXIO I2C

Example application showing FlexIO I2C driver usage

#### Application description

The purpose of this demo application is to show you the usage of the FlexIO I2C driver found on the S32K142 SoC using S32 SDK API.

The application uses FlexIO I2C driver as master to make a send and a receive data request. The slave device for this example is the LPI2C instance, which is configured to act as a bus slave. The setup can't be changed to use FlexIO I2C as slave because this mode is not supported by FlexIO module. The slave and master buffers will be checked after each transfer by the application, user shall check **isTransferOk** variable to see if the transmissions are successful. If transfers is **Ok**, the **LED** on board will turn **Green**, otherwise the **LED** will turn **RED**.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142EVB-Q100 board
- 1 Power Adapter 12V (if the board can't be powered from the USB)
- 1 Personal Computer
- 1 PEMicro Multilink Debugger
- 1 Jlink Lite Debugger

#### Boards supported

The following boards are supported by this application:

- S32K14xCVD-Q064 with S32K-MB
- S32K142EVB-Q100 board

#### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
FLEXIO SDA (PTA11)	J1.2 - J1.1	J9.22 - J9.29
FLEXIO SCL (PTA0)	J5.7 - J1.3	J9.31 - J9.30
LPI2C SDA (PTA2)	J1.1 - J1.2	J9.29 - J9.22
LPI2C SCL (PTA3)	J1.3 - J5.7	J9.30 - J9.31
RED_LED (PTD15)	RGB_RED - wired on board	
GREEN_LED (PTD16)	RGB_GREEN - wired on board	

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **flexio\_i2c\_s32k116**. Then click on **Finish**.

The project should now be copied into you current workspace.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Debugging the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
flexio_i2c_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
flexio_i2c_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

## Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

## 13.4.18 FLEXIO I2S MASTER

Example application showing FlexIO I2S driver usage

### Application description

The purpose of this demo application is to show you the usage of the FlexIO I2S driver found on the S32K142 SoC using S32 SDK API.

The application uses FlexIO I2S driver to make a data transfer of a defined size. The application will work in conjunction with the flexio\_i2s\_slave demo on S32K14x.



The application displays on the host PC terminal a menu in which the user can select to: For Slave board: "Press: 1) [Slave] Send data 2) [Slave] Received data Enter your input:"

For Master board: "Press: 1) [Master] Send data 2) [Master] Received data Enter your input:"

Select Send/Receive on Slave first. After that select Receive/Send on Master.

The slave buffers and master buffers will be checked after each transfer by the application, RED or GREEN led will be lit depend on the check result. Red led will turn if data does not match. Green led will turn if then data is transfered correctly.

The MASTER I2S driver is configured to use DMA for transfers.

Data size is configured by TRANSFER\_SIZE define, by default is configured to be 64 Bytes.

### Prerequisites

To run the example you will need to have the following items:

- 2 S32K142 board
- 2 Power Adapter 12V (if the board can't be powered from the USB)
- 4 Dupont male to male cable
- 1 Personal Computer
- 2 PEmicro Multilink Debugger (optional, users can use J-link)

### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100

### Hardware Wiring

The following connections must be done to for this example application to work: Connect each FlexIO pin board master to pin board slave.

PIN FUNCTION	S32K142EVB-Q100 MASTER	S32K142EVB-Q100 SLAVE
FLEXIO SCK	J6.5 (PTD9)	J1.1(PTA2)
FLEXIO WS	J1.2 (PTA11)	J1.3(PTA3)
FLEXIO MASTER TX - SLAVE RX	J5.7 (PTA0)	J6.19(PTA9)
FLEXIO MASTER RX - SLAVE TX	J5.5 (PTA1)	J6.17(PTA8)
RED_LED (PTD15)	RGB_RED - wired on board	RGB_RED - wired on board
GREEN_LED (PTD16)	RGB_GREEN - wired on board	RGB_GREEN - wired on board
UART	Wired on board	Wired on board

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **flexio\_i2s\_master\_↔s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

## 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

## 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

## 4. Debugging the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
flexio_i2s_master_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
flexio_i2s_master_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### Notes

For this example it is necessary to open a terminal emulator and configure it with:

- 9600 baud
- One stop bit
- No parity
- No flow control
- '\n' line ending

### 13.4.19 FLEXIO I2S SLAVE

Example application showing FlexIO I2S driver usage

#### Application description

The purpose of this demo application is to show you the usage of the FlexIO I2S driver found on the S32K142 SoC using S32 SDK API.

The application uses FlexIO I2S driver to make a data transfer of a defined size. The application will work in conjunction with the flexio\_i2s\_master demo on S32K14x.

The application displays on the host PC terminal a menu in which the user can select to: For Slave board: "Press: 1) [Slave] Send data 2) [Slave] Received data Enter your input:"

For Master board: "Press: 1) [Master] Send data 2) [Master] Received data Enter your input:"

Select Send/Receive on Slave first. After that select Receive/Send on Master.

The slave buffers and master buffers will be checked after each transfer by the application, RED or GREEN led will be lit depend on the check result. Red led will turn if data does not match. Green led will turn if then data is transferred correctly.

The SLAVE I2S driver is configured to use interrupt for transfers.

Data size is configured by TRANSFER\_SIZE define, by default is configured to be 64 Bytes.

#### Prerequisites

To run the example you will need to have the following items:

- 2 S32K142 board
- 2 Power Adapter 12V (if the board can't be powered from the USB)
- 4 Dupont male to male cable
- 1 Personal Computer
- 2 PEMicro Multilink Debugger (optional, users can use J-link)

#### Boards supported

The following boards are supported by this application:

- S32K142EVb-Q100

#### Hardware Wiring

The following connections must be done to for this example application to work: Connect each FlexIO pin board master to pin board slave.

PIN FUNCTION	S32K142EVb-Q100 MASTER	S32K142EVb-Q100 SLAVE
FLEXIO SCK	J6.5 (PTD9)	J1.1(PTA2)
FLEXIO WS	J1.2 (PTA11)	J1.3(PTA3)
FLEXIO MASTER TX - SLAVE RX	J5.7 (PTA0)	J6.19(PTA9)
FLEXIO MASTER RX - SLAVE TX	J5.5 (PTA1)	J6.17(PTA8)
RED_LED (PTD15)	RGB_RED - wired on board	RGB_RED - wired on board
GREEN_LED (PTD16)	RGB_GREEN - wired on board	RGB_GREEN - wired on board
UART	Wired on board	Wired on board

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **flexio\_i2s\_slave\_↔s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

##### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those

will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Debugging the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
flexio_i2s_slave_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
flexio_i2s_slave_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEmicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### Notes

For this example it is necessary to open a terminal emulator and configure it with:

- 9600 baud
- One stop bit
- No parity
- No flow control
- '\n' line ending

### 13.4.20 FLEXIO UART

Example application showing FlexIO UART driver usage

#### Application description

The purpose of this demo application is to show you the usage of the FlexIO UART driver found on the S32K142 SoC using S32 SDK API.

Two instances of the FlexIO UART driver are used to display a welcome message ("Hello World") and then echo the data received from host.

User shall send a string. If the board receives the user's string, then the same string shall be sent again.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board can't be powered from the USB)
- 2 Dupont female to female cable
- 1 Personal Computer
- 1 Jlink Lite Debugger or 1 PEmicro Debugger (optional, users can use Open SDA)
- UART to USB converter if it is not included on the target board. (Please consult your boards documentation to check if UART-USB converter is present).

#### Boards supported

The following boards are supported by this application:

- S32K142EVb-Q100

#### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVb-Q100
FLEXIO_UART RX ( <b>PTA11</b> )	J1.2 - J4.4
FLEXIO_UART TX ( <b>PTA0</b> )	J5.7 - J4.2

#### Note

The application uses on board USB - UART chips to transfer data from board to host PC. Use an USB type B cable to connect to the J16 connector on the mainboard.

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **flexio\_uart\_s32k142**. Then click on **Finish**.  
The project should now be copied into you current workspace.

##### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

##### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

##### 4. Debugging the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<b>flexio_uart_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Segger Jlink debuggers
<b>flexio_uart_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers
<b>flexio_uart_s32k142_debug_ram_pemicro</b>	Debug the RAM configuration using PEMicro debuggers
<b>flexio_uart_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### Notes

For this example it is necessary to open a terminal emulator and configure it with:

- 115200 baud
- One stop bit
- No parity
- No flow control

#### 13.4.21 LPI2C MASTER

Driver example that will show the LPI2C Master functionality

#### Application description

The purpose of this demo application is to show you the usage of the LPI2C module available on the S32K142 MCU as a **master** using S32 SDK.

- The application uses S32 SDK API to initialize the LPI2C module as a master node and in Fast operation speed after configuring the clocks and pins needed to use the I2C. The example sends requests to a slave, found at the configured address, the first being a TX request, while the other being a RX request. Run Slave first, after that Run Master. The master buffers will be checked after each transfer by the application, RED or GREEN led will be turn on or turn off depending on the check result. Red led will turn on if data does not match. Green led will turn on if then data is transfered correctly.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 3 Dupont cables (male to male or female to female depending on the boards)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
LPI2C SCL ( <b>PTA3</b> )	J1-3 - Slave SCL	J9-30 - Slave SCL
LPI2C SDA ( <b>PTA2</b> )	J1-1 - Slave SDA	J9-29 - Slave SDA
GND ( <b>GND</b> )	J3-11 - Slave GND	J6 - Slave GND

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **lpi2c\_master\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace. Wait for the S32 Configuration was initialized and ready.

#### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

#### 3. Building the project

Select the configuration to be built **RAM** (Debug\_RAM) or **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

#### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<b>lpi2c_master_s32k142_debug_ram_pemicro</b>	Debug the RAM configuration using PEMicro debuggers
<b>lpi2c_master_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers
<b>lpi2c_master_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Jlink debuggers
<b>lpi2c_master_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Jlink debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

**Note**

For more detailed information related to S32 Design Studio usage please consult the available documentation.

**13.4.22 LPI2C SLAVE**

Driver example that will show the LPI2C Slave functionality

**Application description**

The purpose of this demo application is to show you the usage of the LPI2C module available on the S32K142 MCU as a **slave** using S32 SDK.

- The application uses S32 SDK API to initialize the LPI2C module as a slave node and in Fast operation speed after configuring the clocks and pins needed to use the I2C. example uses the LPI2C callback to respond to requests such as:
  - data receive
  - data transmit
  - buffer full or empty. Run Slave first, after that Run Master. The slave buffers will be checked after each transfer by the application, RED or GREEN led will be turn on or turn off depending on the check result. Red led will turn on if data does not match. Green led will turn on if then data is transfered correctly.

To check if the transmission is successful the user has to verify that the master rx buffer has the same elements as slave tx buffer and slave rx buffer has the same elements as master tx buffer after running the example.

**Prerequisites**

To run the example you will need to have the following items:

- 1 S32K142 board
- 3 Dupont cables (male to male or female to female depending on the boards)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

**Boards supported**

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

**Hardware Wiring**

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
LPI2C SCL (PTA3)	J1-3 - Master SCL	J9-30 - Master SCL



LPI2C SDA ( <b>PTA2</b> )	J1-1 - Master SDA	J9-29 - Master SDA
GND ( <b>GND</b> )	J3-11 - Master GND	J6 - Master GND

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New -> S32DS Project From** and select **lpi2c\_slave\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace. Wait for the S32 Configuration was initialized and ready.

##### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

##### 3. Building the project

Select the configuration to be built **RAM** (Debug\_RAM) or **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

##### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<b>lpi2c_slave_s32k142_debug_ram_pemicro</b>	Debug the RAM configuration using PEMicro debuggers
<b>lpi2c_slave_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers
<b>lpi2c_slave_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Jlink debuggers
<b>lpi2c_slave_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Jlink debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**. Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.23 LIN MASTER BAREMETAL

Example that shows the usage of the LIN driver in master mode

#### Application description

This example demonstrates the LIN communication between S32K142 EVB Master and Slave using LIN driver without LIN Stack

- A frame contains header and data. The Master node can send header and data, but Slave node only can send data. Base on header, Master node or Slave node will take corresponding action. On Master node:
- Press BUTTON 0:

- For the first time, Master node sends FRAME\_MASTER\_RECEIVE\_DATA header and require slave node responds by sending data (txBuff2).
- For the second time, Master sends FRAME\_SLAVE\_RECEIVE\_DATA header, then continue sending data (txBuff1) and slave node will receive the data.
- If node successful receives data, this node will turn on GREEN\_LED, otherwise turn on RED\_LED.
- Press BUTTON 1:
  - Master node will check current node state. If the state is LIN\_NODE\_STATE\_SLEEP\_MODE, Master node will send wakeup signal and BLUE\_LED will be turned on both nodes, otherwise Master node will send header to set Master node and Slave node to sleep mode and all LED will be turned off both nodes.

### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V
- 4 Dupont female to female cable
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100 RevD
- S32K-MB

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K-MB
BUTTON 0 (PTC12)	SW2 - wired on the board	BTN2 - wired on the board
BUTTON 1 (PTC13)	SW3 - wired on the board	BTN3 - wired on the board
RED_LED (PTD15)	RGB_RED - wired on the board	J12.17 - J11.31
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	J12.16 - J11.30
BLUE_LED (PTD0)	RGB_BLUE - wired on the board	J12.31 - J11.29
GND (GND)	J11-4 - Slave GND	J6 - Slave GND
LIN (*)	J11-1 - Slave LIN	J48.4 - Slave LIN

RX-WORKAROUND | J6-9 (PTD6) - J5-18 (PTC8) | TX-WORKAROUND | J6-11 (PTD7) - J5-16 (PTC9) |

(\*) Those lines must be modulated using a transceiver, if it is not specified the boards already include the LIN transceiver

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **lin\_master\_baremetal\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

## 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**lin\_master\_baremetal\_s32k142**). Select the "ConfigTools" menu then click on the desired configuration (Pins, Clocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

## 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

## 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
lin_master_baremetal_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
lin_master_baremetal_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
lin_master_baremetal_s32k142_debug_ram_pemicro ↩	Debug the RAM configuration using PEmicro debuggers
lin_master_baremetal_s32k142_debug_flash_pemicro ↩	Debug the FLASH configuration using PEmicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### Notes

The default configuration of this example supports S32K142 EVB RevD.

If you want to use S32K-MB please remove EVB macro.

If you want to use other versions of EVB please remove RevD macro and use the Device instance number macro generated by the header file of the LIN module instead of the INST\_LIN\_NUM macro

RX-WORKAROUND and TX-WORKAROUND Pin Function do not need to be set up when using RevD with the Default Configuration of example.

### 13.4.24 LIN SLAVE BAREMETAL

Example that shows the usage of the LIN driver in slave mode

#### Application description

This example demonstrates the LIN communication between S32K142 EVB Master and Slave using LIN driver without LIN Stack

- A frame contains header and data. The Master node can send header and data, but Slave node only can send data. Base on header, Master node or Slave node will take corresponding action.

- If Slave node receives FRAME\_MASTER\_RECEIVE\_DATA header, Slave node will respond by sending data (txBuff2).
- If Slave node receives FRAME\_SLAVE\_RECEIVE\_DATA header, Slave node will receive and check data. If data is success, Slave node will turn on GREEN\_LED, otherwise turn on RED\_LED
- If Slave node receives FRAME\_GO\_TO\_SLEEP header, Slave node will go to sleep mode and turn off all led.
- If Slave node receives a wakeup signal, it will check current node state, if the node state is sleep mode, Slave node will wakeup and turn on BLUE\_LED, otherwise wakeup signal is aborted and keep the previous state.

### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V
- 4 Dupont female to female cable
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K-MB

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K-MB
BUTTON 0 (PTC12)	SW2 - wired on the board	BTN2 - wired on the board
BUTTON 1 (PTC13)	SW3 - wired on the board	BTN3 - wired on the board
RED_LED (PTD15)	RGB_RED - wired on the board	J12.17 - J11.31
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	J12.16 - J11.30
BLUE_LED (PTD0)	RGB_BLUE - wired on the board	J12.31 - J11.29
GND (GND)	J11-4 - Master GND	J6 - Master GND
LIN (*)	J11-1 - Master LIN	J48.4 - Master LIN

RX-WORKAROUND | J6-9 (PTC8) - J5-18 (PTD6) | TX-WORKAROUND | J6-11 (PTC9) - J5-16 (PTD7) |

(\*) Those lines must be modulated using a transceiver, if it is not specified the boards already include the LIN transceiver

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **lin\_slave\_baremetal\_↔ s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

## 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**lin\_slave\_baremetal\_s32k142**). Select the "↔ ConfigTools" menu then click on the desired configuration (Pins, Clocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

## 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

## 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
lin_slave_baremetal_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
lin_slave_baremetal_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
lin_slave_baremetal_s32k142_debug_ram_↔ pemicro	Debug the RAM configuration using PEMicro debuggers
lin_slave_baremetal_s32k142_debug_flash_↔ pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### Notes

The default configuration of this example supports S32K142 EVB RevD.

If you want to use S32K-MB please remove EVB macro.

If you want to use other versions of EVB please remove RevD macro and use the Device instance number macro generated by the header file of the LIN module instead of the INST\_LIN\_NUM macro

RX-WORKAROUND and TX-WORKAROUND Pin Function do not need to be set up when using RevD with the Default Configuration of example.

### 13.4.25 SBC\_UJA116x

Example application using the SBC\_UJA116x driver

#### Application description

The purpose of this demo application is to show you how to use Power modes of SBC\_UJA116x. When application starts SBC is in Normal mode. Press SW2 to change mode to StandBy and SW3 to move to Sleep. Use SW9 button for wake up SBC.

The state of the SBC can be determined based on LED color LED\_GREEN is on in Normal mode otherwise is off. LED\_RED is on in StandBy mode LED\_BLUE is toggle when MCU run and SBC\_FeedWatchdog is called. All LEDs off - SBC in Sleep mode and power supply is off.

## Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

## Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100
LPSP1 CS ( <b>PTD3</b> )	on the board
LPSP1 SCK ( <b>PTD0</b> )	on the board
LPSP1 MOSI ( <b>PTD2</b> )	on the board
LPSP1 MISO ( <b>PTD1</b> )	on the board
Power source	J10 jumper on 1-2 when run application and J10 2-3 when load the application.

## How to run

### 1. Move SBC in FNMC state

pin RSTN (TP506) is held LOW CANH is pulled up to Vbat CANL is pulled down to GND

### 2. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **sbc\_uja116x\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

### 3. Generating the Processor Expert configuration

First go to **Project Explorer** View in S32 DS and select the current project(**sbc\_uja116x\_s32k142**). Then go to **Project** and click on **Generate Processor Expert Code**

Wait for the code generation to be completed before continuing to the next step.

### 4. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 5. Running the project

Go to **Run** and select **Debug Configurations**. There will be two debug configurations for this project:

Configuration Name	Description
sbc_uja116x_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
sbc_uja116x_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### Notes

Test require correct factory settings for example. FNMC bit must be disabled, SBC\_UJA\_SBC\_SDMC\_DIS must be disabled and slpc must be allowed.

#### 13.4.26 System Driver Examples

Applications that show the user how to initialize the communication peripherals

There are currently driver examples with the following modules:

*Click on one of the module to see the available projects*

- [CRC Checksum](#)
- [CSEc key configuration](#)
- [EDMA transfer](#)
- [EIM INJECTION](#)
- [ERM REPORT](#)
- [EWM Interrupt](#)
- [FLASH Partitioning](#)
- [MPU Memory Protect Unit](#)
- [MPU PAL Memory Protection](#)
- [Power Mode Switch](#)
- [Trigger MUX Control](#)
- [WDG PAL Interrupt](#)

- [WDOG Interrupt](#)
- [SECURITY PAL](#)

### 13.4.27 CRC Checksum

Example application showing the usage of the CRC module

#### Application description

The purpose of this demo application is to show you how to use the Cyclic Redundancy Check of the S32K142 with this SDK.

In this example, The CRC is configured to generate a configuration for CCITT standard following:

- CCITT 16 bits standard:

```
{
    .crcWidth = CRC_BITS_16,
    .seed = 0xFFFFU,
    .polynomial = 0x1021U,
    .writeTranspose = CRC_TRANSPOSE_NONE,
    .readTranspose = CRC_TRANSPOSE_NONE,
    .complementChecksum = false
}
```

The application:

1. After reset starts with both GREEN and RED LED turned off.
2. Initializes CRC module with the above CCITT 16 bits standard configuration.
3. Pressing the SW button CRC calculation is initialized with CRC\_data array from input\_data.c file.
4. If the result is correct GREEN LED is turned on. Otherwise RED LED will be turned on.
5. The program stops.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 2 Dupont female to female cables
- 1 Personal Computer
- Debug probe (JLink, PEmicro, OpenSDA)

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB



#### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
LED0 (PTD15/PTC1)	RGB_RED - wired on the board	J12.17 - J11.31
LED1 (PTD16/PTC1)	RGB_GREEN - wired on the board	J12.16 - J11.30
SW (PTC12/PTC12)	SW2 - wired on the board	BUTTON2 - wired on the board

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From** and select **crc\_checksum\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace. Wait for the S32 Configuration was initialized and ready.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<b>crc_checksum_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Segger Jlink debuggers
<b>crc_checksum_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers
<b>crc_checksum_s32k142_debug_ram_pemicro</b>	Debug the RAM configuration using PEMicro debuggers
<b>crc_checksum_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

## Notes

The CRC module in S32K platform supports both big endian and little endian in source data.

## Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### 13.4.28 CSEc key configuration

Basic application that presents basic usecases for the CSEc driver

**Note**

**This example works only for CSEc enabled parts.** SIM\_SDID indicates whether CSEc is available on your device.

**The first time when running the example on the board, or after a key erase, this example should be ran from RAM.**

**The user keys are non-volatile.** Once the key was loaded, in order to update it, the counter should be increased.

**After the user key was loaded using this example, any further full erase of the Flash requires a Challenge-Authentication process.** This can be done by setting the ERASE\_ALL\_KEYS macro to 1.

**After partitioning Flash for CSEc operation, using the JLink Flash configuration of any other project will not work anymore.** Workaround:

- Run csec\_keyconfig example with ERASE\_ALL\_KEYS 1, using PEmicro Flash debug configuration

**Application description**

The purpose of this demo application is to show the user how to use the Cryptographic Services Engine module from the S32K142 MCU with the S32 SDK API.

The implementation demonstrates the following:

- the enablement of the CSEc module, by showing how the Flash should be partitioned (using the Flash driver);
- configuring the MASTER\_ECU key;
- configuring the first user key, using the MASTER\_ECU key as an authorization;
- using the user key for an encryption. In order to update the user key after they were configured using the example, the user should increase the counter used for loading the key. Please note that user should increase counter in order to keep the encryption take its place successfully for 2 cases:
- The user key was already loaded by previous run.
- The example already ran from RAM for CSEc partition. Erasing all the configured keys (including the MASTER\_ECU key) can be done by changing the value of the ERASE\_ALL\_KEYS macro to 1. This will place the part back into factory status (the partition command will need to be run again). Please note that when the Flash is partitioned (the first time running the example on the board, or after a key erase), the example should not be run from Flash (please use the RAM configuration).

**Prerequisites**

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 PEmicro/Jlink Lite Debugger (optional, users can use Open SDA)

**Boards supported**

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
RED_LED (PTD15)	RGB_RED - wired on the board	J12.17 - J11.31
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	J12.16 - J11.30

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From** and select **csec\_keyconfig\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace. Wait for the S32 Configuration was initialized and ready.

##### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

##### 3. Building the project

Select the configuration to be built **RAM** (Debug\_RAM) or **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

##### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
csec_keyconfig_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
csec_keyconfig_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
csec_keyconfig_s32k142_debug_ram_pemicro	Debug the RAM configuration using PEMicro debuggers
csec_keyconfig_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.29 EDMA transfer

Example application showing the usage of the EDMA module

#### Application description

The purpose of this driver example is to show you how to use the eDMA in the following transfer scenarios for the S32K142 MCU using the S32 SDK API.

- Loop memory-to-memory transfer

If the application works correctly, the data shall be transferred correctly to destination memory and a transmission complete interrupt shall be triggered. And the application will not jump to any DEV\_ASSERT.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board can't be powered from the USB)
- 1 Personal Computer
- 1 Jlink Lite Debugger or 1 PEmicro Debugger (optional, users can use Open SDA)

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

#### Hardware Wiring

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **edma\_transfer\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

##### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**edma\_transfer\_s32k142**). Select the "Config↔Tools" menu then click on the desired configuration (Pins, Cocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

##### 3. Building the project

Select the configuration to be built **RAM** (Debug\_RAM) or **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

##### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
--------------------	-------------

<b>edma_transfer_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Segger Jlink debuggers
<b>edma_transfer_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers
<b>edma_transfer_s32k142_debug_ram_pemicro</b>	Debug the RAM configuration using PEMicro debuggers
<b>edma_transfer_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### 13.4.30 EIM INJECTION

Driver example that shows the user how to use the Error Injection Module

#### Application description

The EIM module enables the user to inject 1 bit error or 2 bit errors into bus data, when read from a designated RAM area. The ECC module must correct all 1 bit errors. The ERM module reports any detected memory error. The example runs only on FLASH.

#### Run the code

1. After reset, LED\_RED is turned off, LED\_GREEN is turned on and the value of the test address is initialized.
2. Press button BUTTON0 to initialize the ERM and EIM modules.
3. Read the initialized address; if the value read from the test address is the same as the initialized value, then LED\_GREEN will be turned off and LED\_RED will be turned on.

If application runs success, LED\_GREEN will be turned off and LED\_RED will be turned on after press button 0.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 Jlink Lite Debugger or 1 PEMicro Debugger (optional, users can use Open SDA)

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q144

## Hardware Wiring

The following connections must be done to for this example application to work:



PIN FUNCTION	S32K142EVB-Q144
RED_LED (PTD15)	RGB_RED - wired on the board
GREEN_LED (PTD16)	RGB_GREEN - wired on the board
SW (PTC12)	SW2_BTN0 - wired on the board

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File** -> **New S32DS Project From...** and select **eim\_injection\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

##### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

##### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

##### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be one debug configuration for this project:

Configuration Name	Description
eim_injection_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers
eim_injection_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.31 ERM REPORT

Driver example that shows the user how to use the Error reporting module

#### Application description

The EIM module enables the user to inject 1 bit error or 2 bit errors into bus data, when read from a designated RAM area. The ECC module must correct all 1 bit errors. The ERM module reports any detected memory error. The example runs only on FLASH

### Run the code

1. After reset, LED\_RED is turned off, LED\_GREEN is turned on and the value for address used to test is initialized.
2. Press button SW2 to initialize ERM and EIM modules.
3. Read the address which was initialized, ERM will trigger an interrupt notification which also turns off LED\_GREEN, and turns on the LED\_RED to report a single-bit correction event.
4. Error event details are reported by ERM.

### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V
- 1 Personal Computer
- 1 Jlink Lite Debugger or 1 PEmicro Debugger (optional, users can use Open SDA)

### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100
RED_LED (PTD15)	RGB_RED - wired on the board
GREEN_LED (PTD16)	RGB_GREEN - wired on the board
SW (PTC12)	SW2_BTN0 - wired on the board

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **erm\_report\_s32k142**. Then click on **Finish**.  
The project should now be copied into you current workspace.

#### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

#### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

#### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be one debug configuration for this project:

Configuration Name	Description
<b>erm_report_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers
<b>erm_report_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.  
This Example only run on Flash

### 13.4.32 EWM Interrupt

Driver example that shows the user how to use the External Watchdog Monitor

#### Application description

The purpose of this driver application is to show the user how to use the EWM from the S32K142 using the S32 SDK API.

#### Run the code

1. Turn off LED\_RED and LED\_GREEN.
2. The examples uses the SysTick timer from the ARM core to refresh the EWM counter for 30 times. After each refresh, LED\_RED is also toggled. Within this interval the user can press the button associated with the EWM input pin to assert the interrupt and output pin.
3. After the EWM counter is refreshed 30 times or the user presses the button before refreshing ends, the EWM interrupt is triggered and both LED\_RED and LED\_GREEN are turned ON, then SysTick timer is disabled.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V
- 2 Dupont male to male cable
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100

#### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100
RED_LED (PTD15)	RGB_RED - wired on the board
GREEN_LED (PTD16)	RGB_GREEN - wired on the board
EWM INPUT (PTA3)	J1.3(EWM INPUT) - J2.10(SW2_BTN0)

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **ewm\_interrupt\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

##### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

##### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

##### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
ewm_interrupt_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
ewm_interrupt_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
ewm_interrupt_s32k142_debug_ram_pemicro	Debug the RAM configuration using PEmicro debuggers
ewm_interrupt_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEmicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.33 FLASH Partitioning

Example application which shows the basic operations of the FLASH driver

#### Application description

The purpose of this demo application is to show you the usage of the FLASH driver with the S32 SDK API.

The examples does the following operations:

- Partitions the flash
- Configures FlexNVM region as EEPROM
- Erases flash
- Programs flash
- Write data to EEPROM. Check the status of API which confirms activities of flash module. In addition, user can view value at memory from address 0x3F800 when erases or programs flash. Checks the value at memory from address 0x14000000 when writes data to EEPROM.

#### Note

The FlexNVM memory is partitioned to EEPROM use and is blocked for some erase commands (Erase Sector and Erase Block). As a consequence, loading the program to flash memory may fail on some debuggers. Please perform a mass erase operation on Flash to remove this partitioning after running the example to be able to update your application on target.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)
- 1 micro\_USB cable

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

#### Hardware Wiring

No connections are required for this example.

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **flash\_partitioning\_↔s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace.

##### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
flash_partitioning_s32k142_debug_ram_pemicro	Debug the RAM configuration using PEMicro debuggers
flash_partitioning_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers
flash_partitioning_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
flash_partitioning_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.34 MPU Memory Protect Unit

Basic application that presents the project scenarios for S32 SDK

#### Application description

The purpose of this demo application is to show you how to use the Memory Protection Unit of the s32k142 MCU with this SDK. In this example, MPU regions are configured to have access rights as following:

Region	Core	Debugger	DMA	Address
0	—	rwX	rwX	0x00000000 - 0xFFFFFFFF
1	rwX	rwX	rwX	0x00000000 - 0x0003FEFF
2	-wX	rwX	rwX	0x0003FF00 - 0x0003FF1F
3	r—	rwX	rwX	0x0003FF00 - 0x0003FF1F
4	rwX	rwX	rwX	0x0003FF20 - 0xFFFFFFFF

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

Run the example

1. After reset, MPU will be initialized according to configuration above.
2. Read flash memory at address 0x0003FF04 is permitted.
3. Press button (SW) on the board to ignore read permission by disabling region 3.
4. Read flash memory at address 0x0003FF04 is violated.
5. MPU report the detail of error access on slave port 0 (Crossbar slave port 0 -> Flash Controller).

#### Verification

1. GREEN\_LED on indicate that MPU initialization successful.
2. RED\_LED on (GREEN\_LED off) indicate that there is violated read access reported by MPU.

#### Prerequisites

To run the example you will need to have the following items:

- 1 s32k142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 PEmicro Debugger
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

#### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
RED_LED (PTD15)	RGB_RED - wired on the board	J12.17 - J11.31
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	J12.16 - J11.30
SW (PTC12)	SW2_BTN0 - wired on the board	BUTTON2 - wired on the board

## How to run ##

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **mpu\_memory\_↵ protection\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

#### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

#### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.



#### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<code>mpu_memory_protection_s32k142_debug_ram↔_jlink</code>	Debug the RAM configuration using Segger Jlink debuggers
<code>mpu_memory_protection_s32k142_debug_flash↔_jlink</code>	Debug the FLASH configuration using Segger Jlink debuggers
<code>mpu_memory_protection_s32k142_debug_ram↔_pemicro</code>	Debug the RAM configuration using PEmicro debuggers
<code>mpu_memory_protection_s32k142_debug_flash↔_pemicro</code>	Debug the FLASH configuration using PEmicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### 13.4.35 MPU PAL Memory Protection

Example application that shows how to use the MPU PAL

#### Application description

The purpose of this demo application is to show you how to use the Memory Protection Unit PAL of the S32K142 MCU with this SDK.

In this example, MPU PAL regions are configured to have access rights as following:

Region	Core	Debugger	DMA	Address
0	—	rwX	rwX	0x00000000 - 0xFFFFFFFF
1	rwX	rwX	rwX	0x00000000 - 0x0003FEFF
2	-wX	rwX	rwX	0x0003FF00 - 0x0003FF1F
3	r—	rwX	rwX	0x0003FF00 - 0x0003FF1F
4	rwX	rwX	rwX	0x0003FF20 - 0xFFFFFFFF

Run the example

1. After reset, MPU PAL will be initialized according to configuration above.
2. Read flash memory at address 0x0003FF04 is permitted.
3. Press button (SW) on the board to ignore read permission by disabling region 3.
4. Read flash memory at address 0x0003FF04 is violated.
5. MPU PAL report the detail of error access on slave port 0 (Crossbar slave port 0 -> Flash Controller).

#### Verification

1. GREEN LED on indicate that MPU PAL initialization successful.
2. RED LED on (GREEN LED off) indicate that there is violated read access reported by MPU PAL.

### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 PEmicro Debugger
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
RED_LED (PTD15)	RGB_RED - wired on the board	J12.17 - J11.31
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	J12.16 - J11.30
SW (PTC12)	SW2_BTN0 - wired on the board	BUTTON2 - wired on the board

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **mpu\_pal\_memory\_↔ protection\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

#### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

#### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

#### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<code>mpu_pal_memory_protection_s32k142_debug_ram_jlink</code> ↔	Debug the RAM configuration using Segger Jlink debuggers
<code>mpu_pal_memory_protection_s32k142_debug_flash_jlink</code> ↔	Debug the FLASH configuration using Segger Jlink debuggers
<code>mpu_pal_memory_protection_s32k142_debug_ram_pemicro</code> ↔	Debug the RAM configuration using PEMicro debuggers
<code>mpu_pal_memory_protection_s32k142_debug_flash_pemicro</code> ↔	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### 13.4.36 Power Mode Switch

Example application demonstrating S32K142 power modes

#### Application description

The purpose of the application is to show the user how to enter various power modes of the S32K142 SoC using the S32 SDK API.

The application displays on the host PC terminal a menu in which the user can select to enter:

- High Speed Run (HSRUN)
- Normal Run (RUN)
- Very Low Power Run (VLPR)
- STOP mode 1 (STOP1)
- STOP mode 2 (STOP2)
- Very Low Power Stop (VLPS)

When user selects a mode, PC terminal will show the following text:

Press:

- 1) for HSRUN
- 2) for RUN
- 3) for VLPR
- 4) for STOP1
- 5) for STOP2
- 6) for VLPS

→ Press SW3 to wake up the CPU from STOP1, STOP2 or VLPS mode

Enter your input:

Expected Output:

- If STOP1, STOP2 or VLPS is selected by entering the character: '4', '5' or '6' into PC terminal, LED\_RED will turn on, LED\_GREEN will turn off and the PC terminal will show:

Mode	The content informs
STOP1	***** CPU is going in STOP1 mode...
STOP2	***** CPU is going in STOP2 mode...
VLPS	***** CPU is going in VLPS mode...

- The CPU can be woken up from sleep modes by pressing button SW3 in EVB board, then LED\_RED turn off, LED\_GREEN turn on and PC terminal will show:

Mode	The content informs
STOP1	CPU was entered STOP1 mode successfully and then woke up to exit STOP1 mode.
STOP2	CPU was entered STOP2 mode successfully and then woke up to exit STOP2 mode.
VLPS	CPU was entered VLPS mode successfully and then woke up to exit VLPS mode.

- If user selects HSRUN, RUN or VLPR, the PC terminal will show:

Mode	The content informs
HSRUN	***** CPU is in HSRUN mode ***** Core frequency: 112000000[Hz]
RUN	***** CPU is in RUN mode ***** Core frequency: 48000000[Hz]
VLPR	***** CPU is in VLPR mode ***** Core frequency: 4000000[Hz]

### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVb-Q100
LPUART1 TX (PTC7)	UART_TX - wired on the board
LPUART1 RX (PTC6)	UART_RX - wired on the board
BUTTON (PTC13)	SW3 - wired on the board
LED_0	green led - wired on the board
LED_1	red led - wired on the board

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From** and select **power\_mode\_switch\_↔ s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace. Wait for the S32 Configuration to be initialized and ready.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated.

The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has a blue chip symbol on the top of the toolbar).

In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components.

Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<b>power_mode_switch_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers
<b>power_mode_switch_s32k142_debug_flash_↔ pemicro</b>	Debug the FLASH configuration using PEmicro debuggers.

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

## Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

## Notes

Due to limited RAM size, this example contains only one build configuration for target flash memory (and the associated debug configuration).

For this example it is necessary to open a terminal emulator and configure it with:

- 9600 baud
- One stop bit

- No parity
- No flow control
- '\n' line ending

Clock source is remained in SIRC (8 MHz) before MCU switches from RUN to VLP mode.  
In order to set to default clock for RUN mode. User presses option for RUN or re-initializes clock configuration.

#### 13.4.37 Trigger MUX Control

Example application showing the usage of the TRGMUX module

##### Application description

The purpose of this demo application is to show you how to use the Trigger MUX Control of the S32K14x MCU with this SDK.

The examples use TRGMUX to connect Pin Trigger Mux In10 and LPIT channel 0.

- Initialize TRGMUX with source trigger from TRGMUX\_IN10 and target module is LPIT\_CH0
- Initialize the LPIT Channel 0.
- LED RED on EVB board or LED ORANGE on Motherboard is used to blink led
- Each time when user presses button SW2 on EVB board or SW4 on Motherboard will generate a trigger signal that activates LPIT channel 0 via TRGMUX. After 1ms, LPIT will create an event interrupt and toggle LED

##### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 PEmicro Debugger (optional, users can use Open SDA)

##### Boards supported

The following boards are supported by this application:

- S32K142EVB
- S32K142-MB

##### Hardware Wiring

PIN FUNCTION	S32K142EVB	S32K142-MB
RED_LED(PTD15)	RGB_RED -wired on the board	LED0 (JP49-2 connect J12-18)

SW (PTC11)	SW2_BTN0 (J1-15 connect J2-10)	SW4 - (JP64-2 connect JP64-3; JP63-1 connect JP63-2; JP36-1 connect JP11-22)
------------	--------------------------------	--

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **trgmux\_ipit\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code". Wait for the code generation to be completed before continuing to the next step.

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
trgmux_ipit_s32k142_debug_ram_pemicro	Debug the RAM configuration using PEMicro debuggers
trgmux_ipit_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers
trgmux_ipit_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
trgmux_ipit_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

## Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

## Notes

The TRGMUX module in S32K platform supports both big endian and little endian in source data.

### 13.4.38 WDG PAL Interrupt

Example application that will show the usage of the Watchdog



### Application description

The purpose of this driver application is to show the user how to use the WDG PAL from the S32K142 using the S32 SDK API.

The example uses the SysTick timer from the ARM core to refresh the WDG PAL counter for 30 times. LED0 will toggle when WDG PAL counter is refreshed. After this the WDG PAL counter will expire, WDG PAL interrupt will happen and turn off LED0, LED1. Then the CPU will be reset. If the FLASH configuration will be used, then the program will use the Reset Control Module to detect if the reset was caused by the Watchdog and will stop the execution of the program and turn on LED0, LED1.

### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V
- 1 Personal Computer
- 1 PEmicro Debugger
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
LED0 (PTD15)	RGB_RED - wired on the boards	J12.17 - J11.31
LED1 (PTD16)	RGB_GREEN - wired on the board	J12.16 - J11.30

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **wdg\_pal\_interrupt\_↔s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

#### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code". Wait for the code generation to be completed before continuing to the next step.

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be two debug configurations for this project:

Configuration Name	Description
<b>wdg_pal_interrupt_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers
<b>wdg_pal_interrupt_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### 13.4.39 WDOG Interrupt

Example application that will show the usage of the Watchdog

#### Application description

The purpose of this driver application is to show the user how to use the WDOG from the S32K142 using the S32 SDK API.

The examples uses the SysTick timer from the ARM core to refresh the WDOG counter for 8 times. After this the Watchdog counter will expire and the CPU will be reset. If the FLASH configuration will be used, then the code will use the Reset Control Module to detect if the reset was caused by the Watchdog and will stop the execution of the program.

Run the example on Devkit:

1. After reset, LED 0 and LED 1 is off.
2. Initialize WDOG Interrupt above then LED 0 is toggle 8 times( on 4 times and off 4 times).
3. Watchdog timeout happen then MCU reset and LED 0 and LED 1 is on and The program will stopped.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V
- 1 Personal Computer
- 1 PEMicro
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
RED_LED (PTD15)	RGB_RED - wired on the board	J12.17 - J11.31
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	J12.16 - J11.30

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **wdog\_interrupt\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

#### 2. Generating the S32CT configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

#### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

#### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be two debug configurations for this project:

Configuration Name	Description
<b>wdog_interrupt_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers
<b>wdog_interrupt_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.40 SECURITY PAL

Basic application that presents basic usecases for the Security PAL.

#### Note

**This example works only for CSEc enabled parts.** SIM\_SDID indicates whether CSEc is available on your device.

**The first time when running the example on the board, or after a key erase, this example should be ran from RAM.**

**This example generates a random number.**

**This example demonstrates CBC Encryption/Decryption.**

#### Application description

The purpose of this demo application is to show the user how to use the Security PAL in conjunction with Cryptographic Services Engine module from the S32K14x MCU with the S32 SDK API.

The implementation demonstrates the following:

- the enablement of the Security PAL, used over CSEc module, by showing how the Flash should be partitioned (using the Flash driver);
- initializing the Random Number Generator and generating a vector of 128 random bits;
- configuring the RAM key, with a 128-bit plaintext;
- using the user key for a CBC encryption and a CBC decryption;

If no errors occur during the cryptographic operations, the green LED will be turned on upon completion; if the red LED is lit, the program failed during one of the steps.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 PEmicro/Jlink Lite Debugger (optional, users can use Open SDA)

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **security\_pal\_s32k142**. Select "Copy projects into workspace" and then click on **Finish**.

The project should now be copied into you current workspace.

## 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

## 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

## 4. Running the project

Go to **Run** and select **Debug** Configurations. There will be four debug configurations for this project:

Configuration Name	Description
security_pal_s32k142_debug_flash_jlink Debug_FLASH Jlink	Debug the FLASH configuration using Segger Jlink debuggers
security_pal_s32k142_debug_ram_jlink Debug_RAM Jlink	Debug the RAM configuration using Segger Jlink debuggers
security_pal_s32k142_debug_flash_pemicro Debug_FLASH PEMicro	Debug the FLASH configuration using PEMicro debuggers
security_pal_s32k142_debug_ram_pemicro Debug_RAM PEMicro	Debug the RAM configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### 13.4.41 Timer Driver Examples

Applications that show the user how to initialize the timer peripherals

There are currently driver examples with the following modules:

*Click on one of the module to see the available projects*

- [FTM Combined PWM](#)
- [FTM Periodic Interrupt](#)
- [FTM PWM](#)
- [FTM Signal Measurement](#)
- [IC PAL](#)
- [LPTMR Periodic Interrupt](#)

- [LPTMR Pulse Counter](#)
- [PDB Periodic Interrupt](#)
- [RTC Alarm](#)
- [TIMING PAL](#)
- [PWM PAL](#)
- [OC PAL](#)
- [LPIT Periodic Interrupt](#)

#### 13.4.42 FTM Combined PWM

Example application showing the FTM's combined PWM functionality

##### Application description

The purpose of this demo application is to show you the usage of the Combined PWM mode of the FlexTimer module on S32K142 using S32 SDK API

The example does the following operations:

- Increment or decrement duty cycle
- Update channel duty cycle
- Wait for a number of cycles to make the change visible

Run the example:

1. After run debug, the RED LED and GREEN LED of EVB board will increment or decrement light intensity
2. Use oscilloscope to verify the output signal

##### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

##### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
FTM0 Channel 0 ( <b>PTD15</b> )	RGB_RED - wired on the board	J12.18 - J11.31
FTM0 Channel 1 ( <b>PTD16</b> )	RGB_GREEN - wired on the board	J12.15 - J11.32

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **ftm\_combined\_pwm\_↔s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace.

#### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**ftm\_combined\_pwm\_s32k142**). Right click on the current project -> "S32 Configuration Tool" menu then click on the desired configuration (Pins, Cocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

#### 3. Building the project

Select the configuration **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) to be built by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

#### 4. Running the project

Go to **Run** and select **Debug Configurations**. There are four debug configurations for this project:

Configuration Name	Description
<b>ftm_combined_pwm_s32k142_debug_ram_↔pemicro</b>	Debug the RAM configuration using PEMicro debugger
<b>ftm_combined_pwm_s32k142_debug_flash_↔pemicro</b>	Debug the FLASH configuration using PEMicro debugger
<b>ftm_combined_pwm_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Segger Jlink debugger
<b>ftm_combined_pwm_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debugger

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.43 FTM Periodic Interrupt

Example application showing the FTM's Timer functionality

### Application description

The purpose of this demo application is to show you the usage of the FlexTimer's Timer functionality on S32K142 using S32 SDK API

The example does the following operations:

- Configure FTM0 to generate an interrupt periodically
- The interrupt will blink the LED wired on the board

### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
GPIO PIN	(PTD15) RGB_RED - wired on the board	(PTC0) LED0 - wired on the board

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **ftm\_periodic\_interrupt\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace.

#### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**ftm\_periodic\_interrupt\_s32k142**). Right click on the current project -> "S32 Configuration Tool" menu then click on the desired configuration (Pins, Cocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.



### 3. Building the project

Select the configuration **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) to be built by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There are four debug configurations for this project:

Configuration Name	Description
ftm_periodic_interrupt_s32k142_debug_ram_↔ pemicro	Debug the RAM configuration using PEMicro debuggers
ftm_periodic_interrupt_s32k142_debug_flash_↔ pemicro	Debug the FLASH configuration using PEMicro debuggers
ftm_periodic_interrupt_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
ftm_periodic_interrupt_s32k142_debug_flash_↔ jlink	Debug the FLASH configuration using Segger Jlink debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.44 FTM PWM

Example application showing the FTM's PWM functionality

#### Application description

The purpose of this demo application is to show you the usage of the PWM mode of the FlexTimer module on S32K142 using S32 SDK API

The examples does the following operations:

- Increment or decrement duty cycle
- Update channel duty cycle
- Wait for a number of cycles to make the change visible

Run the example:

1. After run debug, the RED LED of EVB board will increment or decrement light intensity
2. Use oscilloscope to verify the output signal

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

## Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
FTM0 Channel 0 (PTD15)	RGB_RED - wired on the board	J11.31 - J12.18

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **ftm\_pwm\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**ftm\_pwm\_s32k142**). Right click on the current project -> "S32 Configuration Tool" menu then click on the desired configuration (Pins, Cocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

### 3. Building the project

Select the configuration **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) to be built by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There are four debug configurations for this project:

Configuration Name	Description
<b>ftm_pwm_s32k142_debug_ram_pemicro</b>	Debug the RAM configuration using PEMicro debugger
<b>ftm_pwm_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debugger
<b>ftm_pwm_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Segger Jlink debugger
<b>ftm_pwm_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debugger

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

## Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.45 FTM Signal Measurement

Example application showing the FTM's Signal Measurement functionality

##### Application description

The purpose of this demo application is to show you the usage of the FlexTimer's Signal Measurement functionality on S32K142 using S32 SDK API

- The application is configured to generate a PWM signal with a variable frequency which will be measured by another FTM instance. The frequency will range from 1000 Hz to 3000 Hz. Each step changes 100 Hz. The measurement results will be sent to the host PC via LPUART. User is able to compare pwm frequency and measurement frequency.

The pwm frequency must be in measurable frequency range of FTM\_IC. For example, here are the measurable ranges corresponding to the clock source = System clock (48 MHz)

Clock source prescaler	Maximum frequency (Hz)	Minimum frequency (Hz)
1	48,000,000	732.42
2	24,000,000	366.21
4	12,000,000	183.10
8	6,000,000	91.55
16	3,000,000	45.77
32	1,500,000	22.88
64	750,000	11.44
128	375,000	5.72

##### Note

Due to limited RAM size, this example contains only one build configuration for target flash memory (and the associated debug configuration)

##### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)
- 1 microUSB cable

##### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

##### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
FTM0 Out Channel 0 (PTC0)	J4.11 - J2.11	J11.31 - J10.29
FTM1 Input Channel 0 (PTB2)	J2.11 - J4.11	J10.29 - J11.31
LPUART1 TX (PTC7)	UART_TX - wired on the board	J11.26 - J20.2
LPUART1 RX (PTC6)	UART_RX - wired on the board	J11.25 - J20.5

## Notes

For this example it is necessary to open a terminal emulator and configure it with:

- 9600 baud
- One stop bit
- No parity
- No flow control
- '\n' line ending

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **ftm\_signal\_measurement\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**ftm\_signal\_measurement\_s32k142**). Right click on the current project -> "S32 Configuration Tool" menu then click on the desired configuration (Pins, Cocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

### 3. Building the project

Select the configuration **FLASH** (Debug\_FLASH) to be built by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There are two debug configurations for this project:

Configuration Name	Description
<b>ftm_signal_measurement_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debugger
<b>ftm_signal_measurement_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debugger

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

### 5. Output display on Terminal

Welcome message:

This example will show you how to use FTM's signal measurement feature. To achieve that we will generate a modifiable frequency PWM and read it with Input Capture  
Press any key to initiate a new conversion...

#### Expected output:

```
PWM frequency: 1000      Measured frequency: 1000      [Hz]
PWM frequency: 1100      Measured frequency: 1100      [Hz]
PWM frequency: 1200      Measured frequency: 1200      [Hz]
PWM frequency: 1300      Measured frequency: 1300      [Hz]
PWM frequency: 1400      Measured frequency: 1400      [Hz]
PWM frequency: 1500      Measured frequency: 1500      [Hz]
PWM frequency: 1600      Measured frequency: 1600      [Hz]
PWM frequency: 1700      Measured frequency: 1700      [Hz]
PWM frequency: 1800      Measured frequency: 1800      [Hz]
PWM frequency: 1900      Measured frequency: 1900      [Hz]
PWM frequency: 2000      Measured frequency: 2000      [Hz]
PWM frequency: 2100      Measured frequency: 2100      [Hz]
PWM frequency: 2200      Measured frequency: 2200      [Hz]
PWM frequency: 2300      Measured frequency: 2300      [Hz]
PWM frequency: 2400      Measured frequency: 2400      [Hz]
PWM frequency: 2500      Measured frequency: 2500      [Hz]
PWM frequency: 2600      Measured frequency: 2600      [Hz]
PWM frequency: 2700      Measured frequency: 2700      [Hz]
PWM frequency: 2800      Measured frequency: 2800      [Hz]
PWM frequency: 2900      Measured frequency: 2900      [Hz]
PWM frequency: 3000      Measured frequency: 3000      [Hz]
Press any key to initiate a new conversion...
```

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.46 IC PAL

Example application showing the IC's Signal Measurement functionality

#### Application description

The purpose of this demo application is to show you the usage of the IC's Signal Measurement functionality on S32K142 using S32 SDK API

- The application is configured to generate a PWM signal with a variable frequency which will be measured by IC\_PAL. The frequency will range from 1000 Hz to 3000 Hz. Each step changes 100 Hz. The measurement result will be sent to the host PC via LPUART. User is able to compare pwm frequency and measurement frequency

#### Note

Due to limited RAM size, this example contains only one build configuration for target flash memory (and the associated debug configuration)

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)

- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)
- 1 microUSB cable

### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
FTM0 Out Channel 0 ( <b>PTC0</b> )	J4.11 - J2.11	J11.31 - J10.29
FTM1 Input Channel 0 ( <b>PTB2</b> )	J4.11 - J2.11	J11.31 - J10.29
LPUART1 TX ( <b>PTC7</b> )	UART_TX - wired on the board	J11.26 - J20.2
LPUART1 RX ( <b>PTC6</b> )	UART_RX - wired on the board	J11.25 - J20.5

### Notes

For this example it is necessary to open a terminal emulator and configure it with:

- 9600 baud
- One stop bit
- No parity
- No flow control
- '\n' line ending

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **ic\_pal\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace.

#### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**ic\_pal\_s32k142**). Right click on the current project -> "S32 Configuration Tool" menu then click on the desired configuration (Pins, Cocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

#### 3. Building the project

Select the configuration **FLASH** (Debug\_FLASH) to be built by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

#### 4. Running the project

Go to **Run** and select **Debug Configurations**. There are two debug configuration for this project:

Configuration Name	Description
ic_pal_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debugger
ic_pal_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debugger

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

## 5. Output display on Terminal

Welcome message:

```
This example will show you how to use IC's signal measurement feature.
To achieve that we will generate a modifiable frequency PWM and read
it with Input Capture
Press any key to initiate a new conversion...
```

Expected output:

```
PWM frequency: 1000      Measured frequency: 1000      [Hz]
PWM frequency: 1100      Measured frequency: 1100      [Hz]
PWM frequency: 1200      Measured frequency: 1200      [Hz]
PWM frequency: 1300      Measured frequency: 1300      [Hz]
PWM frequency: 1400      Measured frequency: 1400      [Hz]
PWM frequency: 1500      Measured frequency: 1500      [Hz]
PWM frequency: 1600      Measured frequency: 1600      [Hz]
PWM frequency: 1700      Measured frequency: 1700      [Hz]
PWM frequency: 1800      Measured frequency: 1800      [Hz]
PWM frequency: 1900      Measured frequency: 1900      [Hz]
PWM frequency: 2000      Measured frequency: 2000      [Hz]
PWM frequency: 2100      Measured frequency: 2100      [Hz]
PWM frequency: 2200      Measured frequency: 2200      [Hz]
PWM frequency: 2300      Measured frequency: 2300      [Hz]
PWM frequency: 2400      Measured frequency: 2400      [Hz]
PWM frequency: 2500      Measured frequency: 2500      [Hz]
PWM frequency: 2600      Measured frequency: 2600      [Hz]
PWM frequency: 2700      Measured frequency: 2700      [Hz]
PWM frequency: 2800      Measured frequency: 2800      [Hz]
PWM frequency: 2900      Measured frequency: 2900      [Hz]
PWM frequency: 3000      Measured frequency: 3000      [Hz]
Press any key to initiate a new conversion...
```

### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

## 13.4.47 LPTMR Periodic Interrupt

Example application that shows the LPTMR's Timer feature

### Application description

The purpose of this demo application is to show you how to use the LPTMR's Timer functionality from the S32K142 using the S32 SDK API.

- The LPTMR is configured to generate a periodic interrupt at 1 seconds which toggles a LED.

### Prerequisites

To run the example you will need to have the following items:



- 1 S32K142 board
- 1 Power Adapter 12V
- 1 Dupont male to male cable
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100

#### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100
GPIO PIN (PTD0)	RGB_BLUE - wired on the board

#### How to run

##### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **lpit\_periodic\_interrupt\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

##### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**lpit\_periodic\_interrupt\_s32k142**). Select the "ConfigTools" menu then click on the desired configurator (Pins, Cocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

##### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

##### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
lpitmr_periodic_interrupt_s32k142_debug_ram_↔ jlink	Debug the RAM configuration using Segger Jlink debuggers
lpitmr_periodic_interrupt_s32k142_debug_flash_↔ jlink	Debug the FLASH configuration using Segger Jlink debuggers

<b>lptmr_periodic_interrupt_s32k142_debug_ram_↔ pemicro</b>	Debug the RAM configuration using PEMicro debuggers
<b>lptmr_periodic_interrupt_s32k142_debug_flash_↔ _pemicro</b>	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### 13.4.48 LPTMR Pulse Counter

Example application that shows the LPTMR's Pulse Counting feature

#### Application description

The purpose of this demo application is to show you how to use the Low Power Timer's Pulse Counter functionality from the S32K142 using the S32 SDK API.

- The example is configured to trigger an interrupt after three pulses, sourced from one of the board's buttons.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1(2) Dupont male to male cable
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

#### Boards supported

The following boards are supported by this application:

- S32K142EVB-Q100

#### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100
GPIO PIN ( <b>PTD0</b> )	RGB_BLUE - wired on the board
BUTTON ( <b>PTC12</b> )	SW2 - J2.17 - J2.10

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **lptmr\_pulse\_counter\_↵s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**lptmr\_pulse\_counter\_s32k142**). Select the "↵ConfigTools" menu then click on the desired configurator (Pins, Cocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<b>lptmr_pulse_counter_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Segger Jlink debuggers
<b>lptmr_pulse_counter_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers
<b>lptmr_pulse_counter_s32k142_debug_ram_↵pemicro</b>	Debug the RAM configuration using PEMicro debuggers
<b>lptmr_pulse_counter_s32k142_debug_flash_↵pemicro</b>	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

## Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.49 PDB Periodic Interrupt

### Driver example using PDB

#### Application description

The purpose of this demo application is to show you how to use the Programmable Delay Block from the S32K142 using the S32 SDK API.

The PDB is configured to generate a periodic interrupt which toggles a LED.

## See also

adc\_hwtrigger\_group

## Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V
- 1 Personal Computer
- 1 PEmicro Debugger (optional, users can use Open SDA)

## Boards supported

The following boards are supported by this application:

- S32K142EVB
- S32K142-MB

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
GPIO Pin ( <b>PTD15</b> )	RGB_RED - wired on the board	LED0 - J11.31 - J12.18

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File** -> **New S32DS Project From...** and select **pdb\_periodic\_interrupt↔\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code". Wait for the code generation to be completed before continuing to the next step.

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
--------------------	-------------

<b>pdb_periodic_interrupt_s32k142_debug_ram_↔ pemicro</b>	Debug the RAM configuration using PEMicro debuggers
<b>pdb_periodic_interrupt_s32k142_debug_flash_↔ pemicro</b>	Debug the FLASH configuration using PEMicro debuggers
<b>pdb_periodic_interrupt_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Segger Jlink debuggers
<b>pdb_periodic_interrupt_s32k142_debug_flash_↔ jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

### 13.4.50 RTC Alarm

Example application showing basic use cases for the RTC module

#### Application description

The purpose of this demo application is to show you how to use the Real Time Clock module from the S32K142 MCU with the S32 SDK API.

The RTC is configured to generate an interrupt every 1 second toggling GREEN\_LED. If the alarm button is pressed an alarm interrupt toggles the alarm RED\_LED after 5 seconds.

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V
- 2 Dupont male to male cable
- 1 Personal Computer
- 1 PEMicro
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

#### Boards supported

The following boards are supported by this application:

- S32K142-MB
- S32K142EVB-Q100

#### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
RED_LED (PTD15)	RGB_RED - wired on the board	J12.17 - J11.31
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	J12.16 - J11.30
BUTTON (PTC13)	SW3 - wired on the board	SW7 - wired on the board

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From** and select **rtc\_alarm\_s32k142**. Then click on **Finish**.

The project should now be copied into your current workspace. Wait for the S32 Configuration to be initialized and ready.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has a blue chip symbol on the top of the toolbar). In the S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<b>rtc_alarm_s32k142_debug_ram_pemicro</b>	Debug the RAM configuration using PEmicro debuggers
<b>rtc_alarm_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEmicro debuggers
<b>rtc_alarm_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Segger Jlink debuggers
<b>rtc_alarm_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

## Notes

If the example doesn't work, please Flash the Debug\_FLASH configuration and enforce a power on reset of the board.

This is caused by the fact that the register which configures the RTC clock source can only be written once.

**Note**

For more detailed information related to S32 Design Studio usage please consult the available documentation.

**13.4.51 TIMING PAL**

Driver example using TIMING PAL

**Application description**

The purpose of this application is to show you how to use the TIMING PAL over LPIT, LPTMR and FTM timers on the S32K142 using the S32 SDK API.

The application uses one board instance of LPIT, LPTMR and FTM to periodically toggle 3 leds.

**Prerequisites**

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board can't be powered from the USB)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

**Boards supported**

The following boards are supported by this application:

- S32K142EVB-Q100
- S32K142-MB

**Hardware Wiring**

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
BLUE_LED (PTD0)	RGB_BLUE - wired on the board	J11.31 - J12.31
RED_LED (PTD15)	RGB_RED - wired on the board	J11.32 - J12.18
GREEN_LED (PTD16)	RGB_GREEN - wired on the board	J11.29 - J12.15

**How to run****1. Importing the project into the workspace**

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **timing\_pal\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

## 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**timing\_pal\_s32k142**). Select the "ConfigTools" menu then click on the desired configurator (Pins, Cocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

## 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

## 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
timing_pal_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
timing_pal_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
timing_pal_s32k142_debug_ram_pemicro	Debug the RAM configuration using PEMicro debuggers
timing_pal_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

## 13.4.52 PWM PAL

Example application using the PWM PAL

### Application description

The purpose of this demo application is to show you how to use the PWM PAL from the S32K142 CPU using the S32 SDK API. The example will dim the GREEN LED on EVB or ORANGE LED on Motherboard by varying the duty cycle of the PWM signal.

### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 PEmicro Debugger (optional, users can use Open SDA)



### Boards supported

The following boards are supported by this application:

- S32K142-MB
- S32K142EVB-Q100

### Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100	S32K142-MB
FTM0 Channel 0 (PTD16)	RGB_GREEN - wired on the board	LED_ORANGE - Connected J12.15 to JP49.2

### How to run

#### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **pwm\_pal\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

#### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Right click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configuration (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

#### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

#### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<b>pwm_pal_s32k142_debug_ram_pemicro</b>	Debug the RAM configuration using PEMicro debuggers
<b>pwm_pal_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers
<b>pwm_pal_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers
<b>pwm_pal_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Segger Jlink debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

**Note**

For more detailed information related to S32 Design Studio usage please consult the available documentation.

**13.4.53 OC PAL**

Driver example using OC PAL

**Application description**

The purpose of this application is to show you how to use the OC PAL over FTM\_OC on the S32K142 using the S32 SDK API.

This application will toggle green led with period 2 second.

**Prerequisites**

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board can't be powered from the USB)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

**Boards supported**

The following boards are supported by this application:

- S32K142EVB-Q100

**Hardware Wiring**

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100
FTM0 Channel 0 (PTD16)	RGB_GREEN - wired on the board

**How to run****1. Importing the project into the workspace**

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **oc\_pal\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

**2. Generating the S32 configuration**

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. Left click on the current project, then click "Open S32 Configuration" (it has blue chip symbol on the top of the toolbar). In S32 Configuration menu, click on the desired configurator (Pins, Clock, Peripherals, etc.). Clicking on any one of those will generate all the components. Pay attention to any error and warning message. If any configured value is invalid, they will be shown for user. Make the desired changes (if any) then click "Update Code".

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
oc_pal_s32k142_debug_ram_jlink	Debug the RAM configuration using Segger Jlink debuggers
oc_pal_s32k142_debug_flash_jlink	Debug the FLASH configuration using Segger Jlink debuggers
oc_pal_s32k142_debug_ram_pemicro	Debug the RAM configuration using PEMicro debuggers
oc_pal_s32k142_debug_flash_pemicro	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

#### Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

#### 13.4.54 LPIT Periodic Interrupt

Driver example that will show the LPIT functionality

#### Application description

The purpose of this demo application is to show you how to use the Low Power Interrupt Timer from the S32K142 using the S32 SDK API.

- The example is configured to trigger an interrupt every second, which toggles a LED.

#### See also

For other LPIT usage scenario check: ADC\_LOW\_POWER\_group

#### Prerequisites

To run the example you will need to have the following items:

- 1 S32K142 board
- 1 Power Adapter 12V (if the board cannot be powered from the USB port)
- 1 Personal Computer
- 1 Jlink Lite Debugger (optional, users can use Open SDA)

#### Boards supported

The following boards are supported by this application:

- S32K142EVb-Q100

## Hardware Wiring

The following connections must be done to for this example application to work:

PIN FUNCTION	S32K142EVB-Q100
GPIO PIN (PTD0)	RGB_BLUE - wired on the board

## How to run

### 1. Importing the project into the workspace

After opening S32 Design Studio, go to **File -> New S32DS Project From...** and select **lpit\_periodic\_interrupt\_s32k142**. Then click on **Finish**.

The project should now be copied into you current workspace.

### 2. Generating the S32 configuration

The example will run without an active configuration, however if any changes are required, a configuration needs to be generated. The initial configuration will have the same settings as the default example settings. First go to **Project Explorer** View in S32 DS and select the current project(**lpit\_periodic\_interrupt\_s32k142**). Select the "ConfigTools" menu then click on the desired configurator (Pins, Cocks, Peripherals etc...). Clicking on any one of those will generate all the components. Make the desired changes(if any) then click on the "ConfigTools->Update Code" button.

### 3. Building the project

Select the configuration to be built **FLASH** (Debug\_FLASH) or **RAM** (Debug\_RAM) by left clicking on the downward arrow corresponding to the **build** button(. Wait for the build action to be completed before continuing to the next step.

### 4. Running the project

Go to **Run** and select **Debug Configurations**. There will be four debug configurations for this project:

Configuration Name	Description
<b>lpit_periodic_interrupt_s32k142_debug_ram_jlink</b>	Debug the RAM configuration using Segger Jlink debuggers
<b>lpit_periodic_interrupt_s32k142_debug_flash_jlink</b>	Debug the FLASH configuration using Segger Jlink debuggers
<b>lpit_periodic_interrupt_s32k142_debug_ram_pemicro</b>	Debug the RAM configuration using PEMicro debuggers
<b>lpit_periodic_interrupt_s32k142_debug_flash_pemicro</b>	Debug the FLASH configuration using PEMicro debuggers

Select the desired debug configuration and click on **Launch**. Now the perspective will change to the **Debug Perspective**.

Use the controls to control the program flow.

## Note

For more detailed information related to S32 Design Studio usage please consult the available documentation.

## 14 Module Index

### 14.1 Modules

Here is a list of all modules:

**ADC Driver**

**211**

Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL)	229
Automotive Math and Motor Control Library	242
Clock	276
Clock Manager	277
Clock Manager Driver	278
Comparator (CMP)	322
Comparator Driver	326
Controller Area Network - Peripheral Abstraction Layer (CAN PAL)	342
Controller Area Network with Flexible Data Rate (FlexCAN)	359
FlexCAN Driver	440
Cryptographic Services Engine (CSEc)	363
CSEc Driver	255
Cyclic Redundancy Check (CRC)	364
CRC Driver	250
Enhanced Direct Memory Access (eDMA)	411
EDMA Driver	371
Error Injection Module (EIM)	412
EIM Driver	396
Error Reporting Module (ERM)	414
ERM Driver	401
External Watchdog Monitor (EWM)	416
EWM Driver	406
Flash Memory (Flash)	437
Flash Memory (Flash)	417
FlexTimer (FTM)	515
FlexTimer Input Capture Driver (FTM_IC)	553
FlexTimer Module Counter Driver (FTM_MC)	561
FlexTimer Output Compare Driver (FTM_OC)	565
FlexTimer Pulse Width Modulation Driver (FTM_PWM)	571
FlexTimer Quadrature Decoder Driver (FTM_QD)	588
Flexible I/O (FlexIO)	594
FlexIO Common Driver	462

FlexIO I2C Driver	465
FlexIO I2S Driver	474
FlexIO SPI Driver	492
FlexIO UART Driver	506
FreeRTOS	595
I2S - Peripheral Abstraction Layer (I2S PAL)	602
Input Capture - Peripheral Abstraction Layer (IC PAL)	611
Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL)	620
Interrupt Manager (Interrupt)	637
Local Interconnect Network (LIN)	744
LIN Driver	648
LIN Stack	667
Diagnostic services	366
Node configuration	818
Node identification	823
LIN Core API	647
Common Core API.	316
Driver and cluster management	370
Interface management	635
Notification	824
Schedule management	932
Signal interaction	966
User provided call-outs	1060
J2602 Specific API	643
LIN 2.1 Specific API	645
Low level API	752
Transport layer API	1005
Common Transport Layer API	318
Cooked API	361
Initialization	610
Raw API	908
J2602 Transport Layer specific API	644

Node configuration	816
Low Power Inter-Integrated Circuit (LPI2C)	745
LPI2C Driver	670
Low Power Interrupt Timer (LPIT)	746
LPIT Driver	686
Low Power Serial Peripheral Interface (LPSPI)	747
LPSPI Driver	701
Low Power Timer (LPTMR)	750
LPTMR Driver	719
Low Power Universal Asynchronous Receiver-Transmitter (LPUART)	751
LPUART Driver	729
Memory Protection Unit (MPU)	809
MPU Driver	784
Memory Protection Unit Peripheral Abstraction Layer (MPU PAL)	811
MPU PAL	798
OS Interface (OSIF)	825
Output Compare - Peripheral Abstraction Layer (OC PAL)	834
Pins Driver (PINS)	863
PINS Driver	857
Power Manager	865
Power Manager Driver	875
Power_s32k1xx	877
Programmable Delay Block (PDB)	883
PDB Driver	845
Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL)	884
Real Time Clock Driver (RTC)	910
RTC Driver	893
Security Peripheral Abstraction Layer - SECURITY PAL	951
Security PAL	933
Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL)	954
SoC Header file (SoC Header )	967
S32K142 SoC Header file	927

Backward Compatibility Symbols for S32K142	243
Interrupt vector numbers for S32K142	642
Peripheral access layer for S32K142	862
SoC Support	968
S32K142 System Files	928
Structural Core Self Test	973
System Basis Chip Driver (SBC) - UJA116xA Family	978
UJA116xA SBC Driver	1006
System Basis Chip library (SBC) - UJA113x Family	983
00. Library internals	139
Register address map	914
01. Primary control	161
01. Mode handling	159
02. Watchdog	174
03. Fail-safe configuration	178
04. Lock control	193
02. Supply control	173
01. Regulator configuration	162
02. Battery monitor configuration	165
03. SMPS configuration	182
03. Transceiver control	185
01. CAN & CAN partial networking configuration	150
02. LIN configuration	169
04. HVIO bank 0 & 1 control	186
05. Timer control	196
06. Interrupts	200
Bank N fail interrupt enable configuration options (0x38/0x48)	244
Bank N wake-up interrupt enable configuration options (0x37/0x47)	247
Supply interrupt enable configuration options (0x1C)	975
System interrupt enable configuration options (0x04)	984
Transceiver interrupt enable configuration options (0x23)	1003
07. MTPNV and ID	204



HVIO high-side configuration options (0x71)	596
HVIO low-side configuration options (0x72)	599
SBC configuration control options (0x74)	929
Start-up control configuration options (0x73)	970
TRGMUX Driver	985
Timing - Peripheral Abstraction Layer (TIMING PAL)	992
Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL)	1047
Watchdog Peripheral Abstraction Layer (WDG PAL)	1077
WDG PAL	1061
Watchdog timer (WDOG)	1080
WDOG Driver	1068

## 15 Data Structure Index

### 15.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">adc_callback_info_t</a>	Defines a structure used to pass information to the ADC PAL callback	1081
<a href="#">adc_instance_t</a>	Structure storing PAL instance information	1081
<a href="#">can_instance_t</a>	Structure storing PAL instance information	1082
<a href="#">drv_config_t</a>		1082
<a href="#">i2c_instance_t</a>	Structure storing PAL instance information	1083
<a href="#">i2s_instance_t</a>	Structure storing PAL instance information	1084
<a href="#">ic_instance_t</a>	Structure storing PAL instance information	1084
<a href="#">lin_product_id_t</a>	Product id structure Implements : lin_product_id_t_Class	1085
<a href="#">mpu_instance_t</a>	Structure storing PAL instance information	1086
<a href="#">oc_instance_t</a>	Structure storing PAL instance information	1086
<a href="#">oc_pal_state_t</a>	The internal context structure	1087

<a href="#">pwm_instance_t</a>	Structure storing PAL instance information	1087
<a href="#">spi_instance_t</a>	Structure storing PAL instance information	1088
<a href="#">timer_chan_state_t</a>	Runtime state of the Timer channel	1088
<a href="#">timing_instance_t</a>	Structure storing PAL instance information	1089
<a href="#">uart_instance_t</a>	Structure storing PAL instance information	1089
<a href="#">wdg_instance_t</a>	Structure storing PAL instance information	1090

## 16 Module Documentation

### 16.1 00. Library internals

#### 16.1.1 Detailed Description

#### 16.1.2 Internals

This section describes the philosophy of the library, naming conventions and how to start using this library.

##### 16.1.2.1 General description

The SBC incorporates several registers allowing the user to configure its behaviour. Following the SBCs SP↔I protocol, these registers can be read/written to get the status of the device or change the configuration. The [UJA113X\\_SbcTransfer\(\)](#) function is the foundation of this library, which can be used to read/write from/into the SBC. Several functions are available to configure specific functionality of the SBC, however, these just wrap the [UJA113X\\_SbcTransfer\(\)](#) function at the end. Advantage of these functions lies in just providing input parameters and the function takes care of the address, frame length etc.

As it makes no sense to provide different functions just to read a specific register, the [UJA113X\\_SbcTransfer\(\)](#) function should be used in such a case.

Different SBC registers control specific functionality. The number of registers required to control each functionality differs, e.g. the SBC mode is controlled by a single register but to configure CAN, several registers need to be written. From this perspective dedicated functions are available to ease the configuration. Some of these functions configure only one register (e.g. [UJA113X\\_SetWatchdog\(\)](#), [UJA113X\\_SetMode\(\)](#)) and thus bring only little advantage over [UJA113X\\_SbcTransfer\(\)](#), whereas others configure whole group (e.g. [UJA113X\\_SetCan\(\)](#)). It's up to the user which form to use. See the [How to start](#) subsection for details.

Although the library supports connection of multiple UJA113x units to one MCU, the SDK doesn't allow to manipulate these from external user files. Thus the number of SBCs connected to the MCU is hardcoded to 2 in the source file.

Every SBC connected to the host MCU is represented by a structure ([uja113x\\_driver\\_t](#)) holding internal data by the library. An array of such structures is defined in the source code hidden from the user and is used throughout the library. Thus each SBC instance has to be registered to initialize the SBC internal data.

After an SBC has been registered, the user can configure each part of the SBC separately based on instance number and input arguments provided to the [UJA113X\\_SetXYZ](#) functions. Configuration of each functionality is described in separate sections.

In case there is no dedicated function for reading/writing a register, the [UJA113X\\_SbcTransfer\(\)](#) function can be used.

### 16.1.2.2 Naming conventions

This section describes the naming conventions used in the library to ease the navigation in the source code. Although effort has been put to obey these rules, it's not always possible or easy to follow them.

#### Functions

A general naming scheme is used for functions:

**<sbx\_family>\_<action\_name>(input\_parameters)**

For functions configuring specific functionality, the action name always starts with a **Set** keyword as follows:

**<sbx\_family>\_Set<functionality>** (e.g. [UJA113X\\_SetCanPartNet\(\)](#))

#### Structures

A general naming scheme is used for configuration structure:

**<sbx\_family>\_<reg\_fce\_acronym>\_cfg\_t**

The functionality acronym is derived either from a register name or in case of multiple registers dedicated to one functionality by the functionality.

Examples:

[uja113x\\_fsc\\_cfg\\_t](#) -> Fail-Safe Control configuration structure

[uja113x\\_cpnc\\_cfg\\_t](#) -> CAN Partial Networking Control configuration structure

[uja113x\\_dm\\_cfg\\_t](#) -> Data Masks configuration structure

#### Enumerations

A general naming scheme is used for enumeration types:

**<sbx\_family>\_<reg\_acronym>\_<reg\_bit>\_t**

Examples:

[uja113x\\_canc\\_csc\\_t](#) - CAN Control register, bit CSC: CAN Shutdown Control

Naming scheme for enum members:

**<reg\_bit>\_<text>\_<text>...**

Examples:

[CMC\\_TRANSCEIVER\\_OFF](#) - CAN Mode Control: Transceiver offline

[CSC\\_UNDERVOLTAGE](#) - CAN Shutdown Control: Shutdown transceiver during undervoltage

#### Defines

### 16.1.2.3 How to start

The first step before the SBC can be used is to configure the LPSPI peripheral, as the library relies on it. Afterwards the [UJA113X\\_RegisterDevice\(\)](#) should be invoked for every SBC connected to the MCU. While registering a device, important information about the SBC is stored internally. Once a device has been registered, the rest of the functions can be invoked at will to configure the device, read status etc.

#### Step-by-step summary

- 1. Initialize the LPSPI peripheral
- 2. Declare and set the [uja113x\\_drv\\_cfg\\_t](#) structure for every SBC
- 3. Invoke the [UJA113X\\_RegisterDevice\(\)](#) for every SBC
- 4. Configure SBC using the [UJA113X\\_SbcTransfer\(\)](#) function or other library functions

- 5. Change mode to Normal by invoking the [UJA113X\\_SetMode\(\)](#) to run the SBC
- 6. Implement your logic for error/interrupt/status handling

Example:

```
...
...
#define LPSPI_INSTANCE_0    0U    /* SPI instance */
#define LPSPI_TIMEOUT      100U   /* 100 ms timeout */
#define SBC0                0U    /* SBC instance */
...
...
...
void main (void)
{
    /* Init the MCU clock, system related stuff, peripherals etc */
    ...

    /* 1. Initialize the LPSPI peripheral */
    ...

    /* 2. Declare and set the uja113x_drv_cfg_t structure for every SBC */
    uja113x_drv_cfg_t sbcConfig =
    {
        UJA1131HW5V0,
        LPSPI_INSTANCE_0,
        LPSPI_TIMEOUT
    };

    /* 3. Invoke the UJA113X_RegisterDevice() for every SBC */
    UJA113X_RegisterDevice(SBC0, &sbcConfig); /* Mandatory */

    /* 4. Configure SBC using the UJA113X_SbcTransfer() function of other library functions */
    UJA113X_SetWatchdog(...); /* Optional/application specific */
    UJA113X_SbcTransfer(...); /* read back the written register - Optional/application
        specific */
    UJA113X_SetCan(...);      /* Optional/application specific */
    ...
    ...
    UJA113X_SetMtpnv(...);     /* Optional/application specific */
    /* 5. Change mode to Normal by invoking the UJA113X_SetMode() to run the SBC */
    UJA113X_SetMode(...);
}
```

## Modules

- [Register address map](#)

## Data Structures

- struct [uja113x\\_drv\\_cfg\\_t](#)  
SBC driver configuration structure. [More...](#)
- struct [uja113x\\_hvio\\_bank\\_cfg\\_t](#)  
HVIO bank configuration structure. [More...](#)
- struct [uja113x\\_init\\_cfg\\_t](#)  
SBC global initialization structure. [More...](#)
- struct [uja113x\\_driver\\_t](#)  
Internal SBC driver structure  
Implements : [uja113x\\_driver\\_t\\_Class](#). [More...](#)

## Enumerations

- enum [uja113x\\_chip\\_t](#) {  
[UJA1131HW5V0](#) = 0x1100U, [UJA1131HW3V3](#) = 0x1000U, [UJA1132HW5V0](#) = 0x0100U, [UJA1132HW3V3](#) = 0x0000U,  
[UJA1135HW5V0](#) = 0x1101U, [UJA1135HW3V3](#) = 0x1001U, [UJA1136HW5V0](#) = 0x0101U, [UJA1136HW3V3](#) =

```

0x0001U,
UJA1131HWFD5V4 = 0x5100U, UJA1131HWFD3V4 = 0x5000U, UJA1131HWFD5V0 = 0x7100U, UJA1131HWFD3V0 = 0x7000U,
UJA1132HWFD5V4 = 0x4100U, UJA1132HWFD3V4 = 0x4000U, UJA1132HWFD5V0 = 0x6100U, UJA1132HWFD3V0 = 0x6000U }

```

*Driver types.*

## Functions

- `status_t UJA113X_SbcTransfer` (uint32\_t instance, bool read, uint8\_t \*txData, uint8\_t \*rxData, uint8\_t size)  
*Initiates a write/read operation to/from SBC.*
- `status_t UJA113X_Init` (uint32\_t instance, const `uja113x_init_cfg_t` \*const deviceConfig, const `uja113x_drv_cfg_t` \*drvConfig)  
*Initializes complete chip based on input parameters except MTPNV.*
- `status_t UJA113X_RegisterDevice` (uint32\_t instance, const `uja113x_drv_cfg_t` \*drvConfig)  
*Registers SBC internal parameters.*

## Variables

- static `uja113x_driver_t g_drivers` [DRIVER\_COUNT\_\_]  
*Internal array of SBC units connected to MCU.*

### 16.1.3 Data Structure Documentation

#### 16.1.3.1 struct `uja113x_drv_cfg_t`

SBC driver configuration structure.

Every SBC is represented by an internal driver structure, which needs to be initialized first. This is achieved by defining Based on this structure an UJA113X SBC has to be registered.

See also

[UJA113X\\_RegisterDevice\(\)](#)

Implements : `uja113x_drv_cfg_t_Class`

Definition at line 1782 of file `sbc_uja113x.h`.

## Data Fields

- `uja113x_chip_t chipType`  
*Defines the SBC type used by this instance.*
- `uint32_t lpspiInstance`  
*Defines which LPSPI is used to communicate with this device.*
- `uint32_t timeout`  
*Defines the SPI timeout for a single transaction.*

## Field Documentation

#### 16.1.3.1.1 `uja113x_chip_t chipType`

Defines the SBC type used by this instance.

Definition at line 1787 of file `sbc_uja113x.h`.

#### 16.1.3.1.2 uint32\_t lpspiInstance

Defines which LPSPI is used to communicate with this device.

Definition at line 1791 of file sbc\_uja113x.h.

#### 16.1.3.1.3 uint32\_t timeout

Defines the SPI timeout for a single transaction.

Definition at line 1795 of file sbc\_uja113x.h.

#### 16.1.3.2 struct uja113x\_hvio\_bank\_cfg\_t

HVIO bank configuration structure.

Implements : uja113x\_hvio\_bank\_cfg\_t\_Class

Definition at line 1805 of file sbc\_uja113x.h.

#### Data Fields

- [uja113x\\_hvio\\_cfg\\_t hvioConfig](#) [BANK\_HVIO\_CNT]  
*Array of HVIO control register configuration structures (0x30 - 0x33; 0x40 - 0x44)*
- [uja113x\\_bwtc\\_bnwtc\\_t bankWupThr](#)  
*Bank N wake-up threshold configuration options (0x34 / 0x44)*
- uint8\_t [bankWupIntMask](#)  
*Bank N wake-up interrupt masks (0x37 / 0x47)*
- uint8\_t [bankFailIntMask](#)  
*Bank N failure interrupt masks (0x38 / 0x48)*
- [uja113x\\_bscdt\\_cfg\\_t bscdtBankShortCirc](#)  
*Bank N short-circuit detection threshold configuration structure (0x39 / 0x49)*
- [uja113x\\_boldt\\_cfg\\_t boldtBankOpenLoad](#)  
*Bank N open-load detection threshold configuration structure (0x3A / 0x4A)*

#### Field Documentation

##### 16.1.3.2.1 uint8\_t bankFailIntMask

Bank N failure interrupt masks (0x38 / 0x48)

Definition at line 1822 of file sbc\_uja113x.h.

##### 16.1.3.2.2 uint8\_t bankWupIntMask

Bank N wake-up interrupt masks (0x37 / 0x47)

Definition at line 1818 of file sbc\_uja113x.h.

##### 16.1.3.2.3 uja113x\_bwtc\_bnwtc\_t bankWupThr

Bank N wake-up threshold configuration options (0x34 / 0x44)

Definition at line 1814 of file sbc\_uja113x.h.

##### 16.1.3.2.4 uja113x\_boldt\_cfg\_t boldtBankOpenLoad

Bank N open-load detection threshold configuration structure (0x3A / 0x4A)

Definition at line 1830 of file sbc\_uja113x.h.

16.1.3.2.5 [uja113x\\_bscdt\\_cfg\\_t](#) bscdtBankShortCirc

Bank N short-circuit detection threshold configuration structure (0x39 / 0x49)

Definition at line 1826 of file sbc\_uja113x.h.

16.1.3.2.6 [uja113x\\_hvio\\_cfg\\_t](#) hvioConfig[BANK\_HVIO\_CNT]

Array of HVIO control register configuration structures (0x30 - 0x33; 0x40 - 0x44)

Definition at line 1810 of file sbc\_uja113x.h.

16.1.3.3 [struct uja113x\\_init\\_cfg\\_t](#)

SBC global initialization structure.

See also

[UJA113X\\_Init\(\)](#)

Implements : [uja113x\\_init\\_cfg\\_t\\_Class](#)

Definition at line 1841 of file sbc\_uja113x.h.

## Data Fields

- [uja113x\\_wtcd\\_cfg\\_t](#) wdcWatchdogCfg  
*Watchdog configuration.*
- [uja113x\\_mc\\_mc\\_t](#) mcSbcMode  
*Desired mode after initialization.*
- [uja113x\\_fsc\\_cfg\\_t](#) fscCfg  
*Fail-safe configuration.*
- [uint8\\_t](#) sysieMask  
*Defines which system interrupts to enable.*
- [uja113x\\_rc\\_cfg\\_t](#) rcRegulatorCfg  
*Regulator control configuration parameters (0x10)*
- [uja113x\\_bmc\\_cfg\\_t](#) bmcBatMonitorCfg  
*Battery monitor configuration.*
- [uja113x\\_smpsc\\_cfg\\_t](#) smpscSmpsCfg  
*SMPS output voltage control configuration parameter (0x1A)*
- [uint8\\_t](#) supieMask  
*Supply interrupt enable configuration parameters (0x1C)*
- [uja113x\\_canc\\_cfg\\_t](#) cancCanCfg  
*CAN configuration.*
- [uja113x\\_linc\\_cfg\\_t](#) linxLinCtrl  
*LIN 1 and 2 configuration.*
- [uja113x\\_cpnc\\_cfg\\_t](#) cpncCanPartNetCfg  
*CAN partial networking configuration.*
- [uja113x\\_dm\\_cfg\\_t](#) dmCanDataMaskCfg  
*Data mask configuration.*
- [uint8\\_t](#) tieMask  
*Transceiver interrupt enable register configuration parameters (0x23)*
- const [uja113x\\_hvio\\_bank\\_cfg\\_t](#) \* hvioBank0Cfg  
*Configuration of HVIO bank 0 (HVIO 1 - 4)*
- const [uja113x\\_hvio\\_bank\\_cfg\\_t](#) \* hvioBank1Cfg  
*Configuration of HVIO bank 1 (HVIO 5 - 8)*
- const [uja113x\\_timsc\\_cfg\\_t](#) \* timscTimer1Cfg

*Configuration of timer 1.*

- const [uja113x\\_timsc\\_cfg\\_t](#) \* [timscTimer2Cfg](#)

*Configuration of timer 2.*

- const [uja113x\\_timsc\\_cfg\\_t](#) \* [timscTimer3Cfg](#)

*Configuration of timer 3.*

- const [uja113x\\_timsc\\_cfg\\_t](#) \* [timscTimer4Cfg](#)

*Configuration of timer 4.*

- const [uja113x\\_lckc\\_cfg\\_t](#) \* [lckcLockRegCfg](#)

*Configuration of register lock.*

### General purpose memory settings (0x06 - 0x09)

- uint8\_t [memory0](#)
- uint8\_t [memory1](#)
- uint8\_t [memory2](#)
- uint8\_t [memory3](#)

### Field Documentation

#### 16.1.3.3.1 [uja113x\\_bmc\\_cfg\\_t](#) [bmcBatMonitorCfg](#)

Battery monitor configuration.

Definition at line 1874 of file [sbc\\_uja113x.h](#).

#### 16.1.3.3.2 [uja113x\\_canc\\_cfg\\_t](#) [cancCanCfg](#)

CAN configuration.

Definition at line 1886 of file [sbc\\_uja113x.h](#).

#### 16.1.3.3.3 [uja113x\\_cpnc\\_cfg\\_t](#) [cpncCanPartNetCfg](#)

CAN partial networking configuration.

Definition at line 1894 of file [sbc\\_uja113x.h](#).

#### 16.1.3.3.4 [uja113x\\_dm\\_cfg\\_t](#) [dmCanDataMaskCfg](#)

Data mask configuration.

Definition at line 1898 of file [sbc\\_uja113x.h](#).

#### 16.1.3.3.5 [uja113x\\_fsc\\_cfg\\_t](#) [fscCfg](#)

Fail-safe configuration.

Definition at line 1854 of file [sbc\\_uja113x.h](#).

#### 16.1.3.3.6 const [uja113x\\_hvio\\_bank\\_cfg\\_t](#)\* [hvioBank0Cfg](#)

Configuration of HVIO bank 0 (HVIO 1 - 4)

Definition at line 1906 of file [sbc\\_uja113x.h](#).

#### 16.1.3.3.7 const [uja113x\\_hvio\\_bank\\_cfg\\_t](#)\* [hvioBank1Cfg](#)

Configuration of HVIO bank 1 (HVIO 5 - 8)

Definition at line 1910 of file [sbc\\_uja113x.h](#).



**16.1.3.3.8 const uja113x\_lckc\_cfg\_t\* lckcLockRegCfg**

Configuration of register lock.

Definition at line 1930 of file sbc\_uja113x.h.

**16.1.3.3.9 uja113x\_linc\_cfg\_t linxLinCtrl**

LIN 1 and 2 configuration.

Definition at line 1890 of file sbc\_uja113x.h.

**16.1.3.3.10 uja113x\_mc\_mc\_t mcSbcMode**

Desired mode after initialization.

Definition at line 1850 of file sbc\_uja113x.h.

**16.1.3.3.11 uint8\_t memory0**

Definition at line 1862 of file sbc\_uja113x.h.

**16.1.3.3.12 uint8\_t memory1**

Definition at line 1863 of file sbc\_uja113x.h.

**16.1.3.3.13 uint8\_t memory2**

Definition at line 1864 of file sbc\_uja113x.h.

**16.1.3.3.14 uint8\_t memory3**

Definition at line 1865 of file sbc\_uja113x.h.

**16.1.3.3.15 uja113x\_rc\_cfg\_t rcRegulatorCfg**

Regulator control configuration parameters (0x10)

Definition at line 1870 of file sbc\_uja113x.h.

**16.1.3.3.16 uja113x\_smpsc\_cfg\_t smpscSmpsCfg**

SMPS output voltage control configuration parameter (0x1A)

Definition at line 1878 of file sbc\_uja113x.h.

**16.1.3.3.17 uint8\_t supieMask**

Supply interrupt enable configuration parameters (0x1C)

Definition at line 1882 of file sbc\_uja113x.h.

**16.1.3.3.18 uint8\_t sysieMask**

Defines which system interrupts to enable.

Definition at line 1858 of file sbc\_uja113x.h.

**16.1.3.3.19 uint8\_t tieMask**

Transceiver interrupt enable register configuration parameters (0x23)

Definition at line 1902 of file sbc\_uja113x.h.

#### 16.1.3.3.20 `const uja113x_timsc_cfg_t* timscTimer1Cfg`

Configuration of timer 1.

Definition at line 1914 of file `sbc_uja113x.h`.

#### 16.1.3.3.21 `const uja113x_timsc_cfg_t* timscTimer2Cfg`

Configuration of timer 2.

Definition at line 1918 of file `sbc_uja113x.h`.

#### 16.1.3.3.22 `const uja113x_timsc_cfg_t* timscTimer3Cfg`

Configuration of timer 3.

Definition at line 1922 of file `sbc_uja113x.h`.

#### 16.1.3.3.23 `const uja113x_timsc_cfg_t* timscTimer4Cfg`

Configuration of timer 4.

Definition at line 1926 of file `sbc_uja113x.h`.

#### 16.1.3.3.24 `uja113x_wtdc_cfg_t wdcWatchdogCfg`

Watchdog configuration.

Definition at line 1846 of file `sbc_uja113x.h`.

#### 16.1.3.4 `struct uja113x_driver_t`

Internal SBC driver structure

Implements : `uja113x_driver_t_Class`.

Definition at line 64 of file `sbc_uja113x.c`.

##### Data Fields

- [`uja113x\_chip\_t chipType`](#)  
*Chip ID.*
- [`uint32\_t lpspiInstance`](#)  
*LPSPi used to communicate with SBC.*
- [`uint32\_t timeout`](#)  
*SPI timeout.*

##### Field Documentation

#### 16.1.3.4.1 `uja113x_chip_t chipType`

Chip ID.

Definition at line 69 of file `sbc_uja113x.c`.

#### 16.1.3.4.2 `uint32_t lpspiInstance`

LPSPi used to communicate with SBC.

Definition at line 73 of file `sbc_uja113x.c`.

#### 16.1.3.4.3 `uint32_t timeout`

SPI timeout.

Definition at line 77 of file `sbc_uja113x.c`.

## 16.1.4 Enumeration Type Documentation

## 16.1.4.1 enum uja113x\_chip\_t

Driver types.

Implements : uja113x\_chip\_t\_Class

## Enumerator

***UJA1131HW5V0***  
***UJA1131HW3V3***  
***UJA1132HW5V0***  
***UJA1132HW3V3***  
***UJA1135HW5V0***  
***UJA1135HW3V3***  
***UJA1136HW5V0***  
***UJA1136HW3V3***  
***UJA1131HWFD5V4***  
***UJA1131HWFD3V4***  
***UJA1131HWFD5V0***  
***UJA1131HWFD3V0***  
***UJA1132HWFD5V4***  
***UJA1132HWFD3V4***  
***UJA1132HWFD5V0***  
***UJA1132HWFD3V0***

Definition at line 43 of file sbc\_uja113x.h.

## 16.1.5 Function Documentation

## 16.1.5.1 status\_t UJA113X\_Init ( uint32\_t instance, const uja113x\_init\_cfg\_t \*const deviceConfig, const uja113x\_drv\_cfg\_t \* drvConfig )

Initializes complete chip based on input parameters except MTPNV.

## Parameters

in	<i>instance</i>	Index of SBC to configure
in	<i>deviceConfig</i>	SBC configuration structure
in	<i>drvConfig</i>	Driver internal data

## Returns

Error or success status returned by API

Definition at line 148 of file sbc\_uja113x.c.

## 16.1.5.2 status\_t UJA113X\_RegisterDevice ( uint32\_t instance, const uja113x\_drv\_cfg\_t \* drvConfig )

Registers SBC internal parameters.

The library allows use of multiple UJA113X SBCs to be connected to an MCU. From this perspective every SBC has a set of internal data necessary for proper functionality. This function should be used to set these parameters for a single SBC based on input parameters.

**Parameters**

in	<i>instance</i>	SBC index (must be lower than UJA113X_DRIVER_COUNT)
in	<i>drvConfig</i>	SBC configuration structure

**Returns**

SBC\_SUCCESS Configuration successful SBC\_ERR\_NA SBC instance out of range

Definition at line 366 of file sbc\_uja113x.c.

16.1.5.3 **status\_t UJA113X\_SbcTransfer ( uint32\_t instance, bool read, uint8\_t \* txData, uint8\_t \* rxData, uint8\_t size )**

Initiates a write/read operation to/from SBC.

Based on input arguments the function initiates a data transfer with SBC. Although the function allows to transfer any amount of data, the minimum number of bytes to transfer is 2 (address + 1 byte), maximum is 4 (address + 3 bytes) as defined by the SBCs SPI protocol. Otherwise the frame is considered as invalid and an SPI failure is raised by the SBC. The read parameter defines, whether a read or write from/to SBC is requested.

The first byte of txData must contain the address to read/write from/to.

**Parameters**

in	<i>instance</i>	Index of SBC to configure
in	<i>read</i>	Read operation is carried out on true, write otherwise
in	<i>txData</i>	Buffer to write into SBC
out	<i>rxData</i>	Buffer for received data
in	<i>size</i>	Number of bytes to write/read

**Returns**

STATUS\_SUCCESS Write/read successful STATUS\_BUSY SPI peripheral busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

**Warning**

The first byte of txData must contain the register address to read/write from/to!

The function doesn't check the size of data. Make sure you write 2 - 4 bytes!

**Note**

The LSB bit of the address in first byte is not taken into account. The read parameter determines, whether a read/write will be performed.

**See also**

[Register address map](#)

Definition at line 106 of file sbc\_uja113x.c.

**16.1.6 Variable Documentation**

16.1.6.1 **uja113x\_driver\_t g\_drivers[DRIVER\_COUNT\_] [static]**

Internal array of SBC units connected to MCU.

The count of SBC units can be influenced by UJA113X\_DRIVER\_COUNT which can be defined by the user in sbc\_uja113x\_usr.h file. If not defined, the default value is 1!

Implements : multiple\_sbc\_support\_Class

Definition at line 101 of file sbc\_uja113x.c.

## 16.2 01. CAN & CAN partial networking configuration

### 16.2.1 Detailed Description

#### 16.2.2 Introduction

Except configuring mode and shutdown control during over-/undervoltages, the transceiver also supports CAN partial networking (CAN PN) (only devices with CAN partial networking feature). The PN feature allows to wake-up only a single node connected to a bus. From this perspective, several parameters (ID, ID mask, data, data masks etc.) need to be configured to be able to wake up on a dedicated CAN wake-up frame.

##### 16.2.2.1 CAN Partial networking

To enable the CAN PN functionality and confirm its configuration, the CPNC & PNCOK bit in 0x20 register has to be written. The write of PNCOK bit should be done after the CAN PN registers were written, otherwise the PNCOK gets cleared on any write access into registers related to the CAN PN. Thus it's advised to write the CAN PN registers first ([UJA113X\\_SetCanPartNet\(\)](#) & [UJA113X\\_SetDataMasks\(\)](#)). The 0x20 register should be configured afterwards ([UJA113X\\_SetCan\(\)](#)).

In case the CAN PN wake-up frame should not rely on data, the CAN data masks configuration can be skipped. Thus the CAN PN configuration is split in two parts.

#### Data Structures

- struct [uja113x\\_canc\\_cfg\\_t](#)  
*CAN control register configuration structure (0x20) [More...](#)*
- struct [uja113x\\_cpnc\\_cfg\\_t](#)  
*CAN partial networking configuration structure (0x26 - 0x2F) [More...](#)*
- struct [uja113x\\_dm\\_cfg\\_t](#)  
*CAN partial networking data mask registers configuration structure (0x68 - 0x6F) [More...](#)*

#### Typedefs

- typedef bool [uja113x\\_canc\\_cfdc\\_t](#)  
*CAN FD control (0x20:CFDC[6])*
- typedef bool [uja113x\\_canc\\_cpnc\\_t](#)  
*CAN partial networking control parameter (0x20:CPNC[4])*
- typedef uint32\_t [uja113x\\_id\\_id\\_t](#)  
*CAN partial networking ID (0x27 - 0x2A)*
- typedef uint32\_t [uja113x\\_idm\\_idm\\_t](#)  
*CAN partial networking ID mask (0x2B - 0x2E)*
- typedef bool [uja113x\\_fc\\_ide\\_t](#)  
*CAN partial networking ID format options (0x2F:IDE[7])*
- typedef bool [uja113x\\_fc\\_pndm\\_t](#)  
*CAN partial networking data mask options (0x2F:PNDM[6])*

#### Enumerations

- enum [uja113x\\_canc\\_csc\\_t](#) { [CSC\\_NONE](#) = 0x00U, [CSC\\_UNDRVOLTAGE](#) = 0x04U, [CSC\\_OVERVOLTAGE](#) = 0x08U, [CSC\\_UNDRV\\_OVERV](#) = 0x0CU }  
*CAN shut-down control configuration options (0x20:CSC[3:2])*
- enum [uja113x\\_canc\\_cmc\\_t](#) { [CMC\\_TRANSCEIVER\\_OFF](#) = 0x00U, [CMC\\_TRANSCEIVER\\_ACTIVE](#) = 0x01U, [CMC\\_TRANSCEIVER\\_ACTIVE2](#) = 0x02U, [CMC\\_TRANSCEIVER\\_LISTEN](#) = 0x03U }  
*CAN transceiver operating mode configuration options (0x20:CMC[1:0])*

- enum [uja113x\\_dr\\_cdr\\_t](#) {  
[CDR\\_DATA\\_RATE\\_50](#) = 0x00U, [CDR\\_DATA\\_RATE\\_100](#) = 0x01U, [CDR\\_DATA\\_RATE\\_125](#) = 0x02U, [CDR\\_DATA\\_RATE\\_250](#) = 0x03U,  
[CDR\\_DATA\\_RATE\\_500a](#) = 0x04U, [CDR\\_DATA\\_RATE\\_500b](#) = 0x05U, [CDR\\_DATA\\_RATE\\_500c](#) = 0x06U,  
[CDR\\_DATA\\_RATE\\_1000](#) = 0x07U }

*CAN data rate configuration options (0x26:CDR[2:0])*

- enum [uja113x\\_fc\\_dlc\\_t](#) {  
[DLC\\_0](#) = 0x00U, [DLC\\_1](#) = 0x01U, [DLC\\_2](#) = 0x02U, [DLC\\_3](#) = 0x03U,  
[DLC\\_4](#) = 0x04U, [DLC\\_5](#) = 0x05U, [DLC\\_6](#) = 0x06U, [DLC\\_7](#) = 0x07U,  
[DLC\\_8](#) = 0x08U }

*Number of data bytes in a CAN wake-up frame options (0x2F:DLC[3:0])*

## Functions

- status\_t [UJA113X\\_SetCan](#) (uint32\_t instance, const [uja113x\\_canc\\_cfg\\_t](#) \*cancConfig)  
*Sets the CAN control register (0x20) based on input configuration object.*
- status\_t [UJA113X\\_SetCanPartNet](#) (uint32\_t instance, const [uja113x\\_cpnc\\_cfg\\_t](#) \*cpncConfig)  
*Sets the CAN partial networking control register (0x26 - 0x2F) based on input configuration object.*
- status\_t [UJA113X\\_SetDataMasks](#) (uint32\_t instance, const [uja113x\\_dm\\_cfg\\_t](#) \*dmConfig)  
*Sets the CAN partial networking data mask registers (0x68 - 0x6F) based on input configuration object.*

## 16.2.3 Data Structure Documentation

### 16.2.3.1 struct [uja113x\\_canc\\_cfg\\_t](#)

CAN control register configuration structure (0x20)

Implements : [uja113x\\_canc\\_cfg\\_t\\_Class](#)

Definition at line 1538 of file [sbc\\_uja113x.h](#).

#### Data Fields

- [uja113x\\_canc\\_cfdc\\_t](#) [canFDCtrl](#)  
*Configures whether CAN FD frames are tolerated or not (0x20:CFDC[6])*
- [uja113x\\_canc\\_cpnc\\_t](#) [canPartNetCtrl](#)  
*Disables/enables CAN selective wake-up (0x20:CPNC[4])*
- [uja113x\\_canc\\_csc\\_t](#) [canShutdownCtrl](#)  
*Configures CAN shut-down behaviour during battery over- or undervoltage (0x20:CSC[3:2])*
- [uja113x\\_canc\\_cmc\\_t](#) [canModeCtrl](#)  
*Selects the CAN transceiver operating mode (0x20:CMC[1:0])*

## Field Documentation

### 16.2.3.1.1 [uja113x\\_canc\\_cfdc\\_t](#) [canFDCtrl](#)

Configures whether CAN FD frames are tolerated or not (0x20:CFDC[6])

Definition at line 1543 of file [sbc\\_uja113x.h](#).

### 16.2.3.1.2 [uja113x\\_canc\\_cmc\\_t](#) [canModeCtrl](#)

Selects the CAN transceiver operating mode (0x20:CMC[1:0])

Definition at line 1555 of file [sbc\\_uja113x.h](#).

**16.2.3.1.3 uja113x\_canc\_cpnc\_t canPartNetCtrl**

Disables/enables CAN selective wake-up (0x20:CPNC[4])

Definition at line 1547 of file sbc\_uja113x.h.

**16.2.3.1.4 uja113x\_canc\_csc\_t canShutdownCtrl**

Configures CAN shut-down behaviour during battery over- or undervoltage (0x20:CSC[3:2])

Definition at line 1551 of file sbc\_uja113x.h.

**16.2.3.2 struct uja113x\_cpnc\_cfg\_t**

CAN partial networking configuration structure (0x26 - 0x2F)

Implements : uja113x\_cpnc\_cfg\_t\_Class

Definition at line 1592 of file sbc\_uja113x.h.

**Data Fields**

- [uja113x\\_dr\\_cdr\\_t canDataRate](#)  
*Configures CAN data rate (0x26:CDR[2:0])*
- [uja113x\\_id\\_id\\_t id](#)  
*Configures CAN wake-up ID (0x27 - 0x2A)*
- [uja113x\\_idm\\_idm\\_t idMask](#)  
*Configures CAN wake-up ID mask (0x2B - 0x2E)*
- [uja113x\\_fc\\_ide\\_t ide](#)  
*Configures whether standard/extended ID is used for wake-up (0x2F:IDE[7])*
- [uja113x\\_fc\\_pndm\\_t partNetDataMask](#)  
*Configures whether DLC and data are checked during wake-up (0x2F:PNDM[6])*
- [uja113x\\_fc\\_dlc\\_t dataLenCode](#)  
*Configures the number of expected bytes during wake-up (0x2F:DLC[3:0])*

**Field Documentation****16.2.3.2.1 uja113x\_dr\_cdr\_t canDataRate**

Configures CAN data rate (0x26:CDR[2:0])

Definition at line 1597 of file sbc\_uja113x.h.

**16.2.3.2.2 uja113x\_fc\_dlc\_t dataLenCode**

Configures the number of expected bytes during wake-up (0x2F:DLC[3:0])

Definition at line 1617 of file sbc\_uja113x.h.

**16.2.3.2.3 uja113x\_id\_id\_t id**

Configures CAN wake-up ID (0x27 - 0x2A)

Definition at line 1601 of file sbc\_uja113x.h.

**16.2.3.2.4 uja113x\_fc\_ide\_t ide**

Configures whether standard/extended ID is used for wake-up (0x2F:IDE[7])

Definition at line 1609 of file sbc\_uja113x.h.

#### 16.2.3.2.5 uja113x\_idm\_idm\_t idMask

Configures CAN wake-up ID mask (0x2B - 0x2E)

Definition at line 1605 of file sbc\_uja113x.h.

#### 16.2.3.2.6 uja113x\_fc\_pndm\_t partNetDataMask

Configures whether DLC and data are checked during wake-up (0x2F:PNDM[6])

Definition at line 1613 of file sbc\_uja113x.h.

#### 16.2.3.3 struct uja113x\_dm\_cfg\_t

CAN partial networking data mask registers configuration structure (0x68 - 0x6F)

Implements : uja113x\_dm\_cfg\_t\_Class

Definition at line 1627 of file sbc\_uja113x.h.

##### Data Fields

- [uint8\\_t dm0](#)  
*Data mask 0 configuration (0x68:DM0[7:0])*
- [uint8\\_t dm1](#)  
*Data mask 1 configuration (0x69:DM1[7:0])*
- [uint8\\_t dm2](#)  
*Data mask 2 configuration (0x6A:DM2[7:0])*
- [uint8\\_t dm3](#)  
*Data mask 3 configuration (0x6B:DM3[7:0])*
- [uint8\\_t dm4](#)  
*Data mask 4 configuration (0x6C:DM4[7:0])*
- [uint8\\_t dm5](#)  
*Data mask 5 configuration (0x6D:DM5[7:0])*
- [uint8\\_t dm6](#)  
*Data mask 6 configuration (0x6E:DM6[7:0])*
- [uint8\\_t dm7](#)  
*Data mask 7 configuration (0x6F:DM7[7:0])*

##### Field Documentation

#### 16.2.3.3.1 uint8\_t dm0

Data mask 0 configuration (0x68:DM0[7:0])

Definition at line 1629 of file sbc\_uja113x.h.

#### 16.2.3.3.2 uint8\_t dm1

Data mask 1 configuration (0x69:DM1[7:0])

Definition at line 1630 of file sbc\_uja113x.h.

#### 16.2.3.3.3 uint8\_t dm2

Data mask 2 configuration (0x6A:DM2[7:0])

Definition at line 1631 of file sbc\_uja113x.h.



#### 16.2.3.3.4 uint8\_t dm3

Data mask 3 configuration (0x6B:DM3[7:0])

Definition at line 1632 of file sbc\_uja113x.h.

#### 16.2.3.3.5 uint8\_t dm4

Data mask 4 configuration (0x6C:DM4[7:0])

Definition at line 1633 of file sbc\_uja113x.h.

#### 16.2.3.3.6 uint8\_t dm5

Data mask 5 configuration (0x6D:DM5[7:0])

Definition at line 1634 of file sbc\_uja113x.h.

#### 16.2.3.3.7 uint8\_t dm6

Data mask 6 configuration (0x6E:DM6[7:0])

Definition at line 1635 of file sbc\_uja113x.h.

#### 16.2.3.3.8 uint8\_t dm7

Data mask 7 configuration (0x6F:DM7[7:0])

Definition at line 1636 of file sbc\_uja113x.h.

### 16.2.4 Typedef Documentation

#### 16.2.4.1 typedef bool uja113x\_canc\_cfdc\_t

CAN FD control (0x20:CFDC[6])

false - CAN FD tolerance disabled, the CAN FD frames are detected as erroneous while in sleep with partial networking enabled

true - CAN FD tolerance enabled, the can FD frames are ignored while in sleep with partial networking enabled

Implements : uja113x\_canc\_cfdc\_t\_Class

Definition at line 586 of file sbc\_uja113x.h.

#### 16.2.4.2 typedef bool uja113x\_canc\_cpnc\_t

CAN partial networking control parameter (0x20:CPNC[4])

false - Disable CAN selective wake-up

true - Enable CAN selective wake-up

Implements : uja113x\_canc\_cpnc\_t\_Class

Definition at line 597 of file sbc\_uja113x.h.

#### 16.2.4.3 typedef bool uja113x\_fc\_ide\_t

CAN partial networking ID format options (0x2F:IDE[7])

false - standard frame format (11-bit)

true - extended frame format (11-bit)

Implements : uja113x\_fc\_ide\_t\_Class

Definition at line 799 of file sbc\_uja113x.h.

#### 16.2.4.4 typedef bool uja113x\_fc\_pndm\_t

CAN partial networking data mask options (0x2F:PNDM[6])

false - data length code and data field are ignored for wake-up

true - data length code and data field are evaluated at wake-up

Implements : uja113x\_fc\_pndm\_t\_Class

Definition at line 810 of file sbc\_uja113x.h.

#### 16.2.4.5 typedef uint32\_t uja113x\_id\_id\_t

CAN partial networking ID (0x27 - 0x2A)

Implements : uja113x\_id\_id\_t\_Class

Definition at line 779 of file sbc\_uja113x.h.

#### 16.2.4.6 typedef uint32\_t uja113x\_idm\_idm\_t

CAN partial networking ID mask (0x2B - 0x2E)

Implements : uja113x\_idm\_idm\_t\_Class

Definition at line 788 of file sbc\_uja113x.h.

### 16.2.5 Enumeration Type Documentation

#### 16.2.5.1 enum uja113x\_canc\_cmc\_t

CAN transceiver operating mode configuration options (0x20:CMC[1:0])

Implements : uja113x\_canc\_cmc\_t\_Class

##### Enumerator

**CMC\_TRANSCEIVER\_OFF** Offline/Offline bias mode.

**CMC\_TRANSCEIVER\_ACTIVE** Active mode (when the SBC is in normal mode).

**CMC\_TRANSCEIVER\_ACTIVE2** Active mode (when the SBC is in normal mode); VCAN undervoltage disabled.

**CMC\_TRANSCEIVER\_LISTEN** Listen-only mode.

Definition at line 628 of file sbc\_uja113x.h.

#### 16.2.5.2 enum uja113x\_canc\_csc\_t

CAN shut-down control configuration options (0x20:CSC[3:2])

Implements : uja113x\_canc\_csc\_t\_Class

##### Enumerator

**CSC\_NONE** CAN transceiver is not shut down when a battery monitor under- or overvoltage interrupt is generated.

**CSC\_UNDRVOLTAGE** CAN transceiver shuts down in response to a battery monitor undervoltage (BMUI) interrupt (SBC in normal mode).

**CSC\_OVERVOLTAGE** CAN transceiver shuts down in response to a battery monitor overvoltage (BMOI) interrupt (SBC in normal mode).

**CSC\_UNDRV\_OVERV** CAN transceiver shuts down in response to BMUI or BMOI interrupt (SBC in normal mode).

Definition at line 606 of file sbc\_uja113x.h.

## 16.2.5.3 enum uja113x\_dr\_cdr\_t

CAN data rate configuration options (0x26:CDR[2:0])

Implements : uja113x\_dr\_cdr\_t\_Class

## Enumerator

**CDR\_DATA\_RATE\_50** 50 kbit/s  
**CDR\_DATA\_RATE\_100** 100 kbit/s  
**CDR\_DATA\_RATE\_125** 125 kbit/s  
**CDR\_DATA\_RATE\_250** 250 kbit/s  
**CDR\_DATA\_RATE\_500a** reserved (intended for future use; currently selects 500 kbit/s)  
**CDR\_DATA\_RATE\_500b** 500 kbit/s  
**CDR\_DATA\_RATE\_500c** reserved (intended for future use; currently selects 500 kbit/s)  
**CDR\_DATA\_RATE\_1000** 1000 kbit/s

Definition at line 758 of file sbc\_uja113x.h.

## 16.2.5.4 enum uja113x\_fc\_dlc\_t

Number of data bytes in a CAN wake-up frame options (0x2F:DLC[3:0])

Implements : uja113x\_fc\_dlc\_t\_Class

## Enumerator

**DLC\_0** 0 bytes expected  
**DLC\_1** 1 bytes expected  
**DLC\_2** 2 bytes expected  
**DLC\_3** 3 bytes expected  
**DLC\_4** 4 bytes expected  
**DLC\_5** 5 bytes expected  
**DLC\_6** 6 bytes expected  
**DLC\_7** 7 bytes expected  
**DLC\_8** 8 bytes expected

Definition at line 819 of file sbc\_uja113x.h.

## 16.2.6 Function Documentation

## 16.2.6.1 status\_t UJA113X\_SetCan ( uint32\_t instance, const uja113x\_canc\_cfg\_t \* canConfig )

Sets the CAN control register (0x20) based on input configuration object.

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the CAN control register has not been written properly and the process has to be repeated.

This function configures the PNCOK bit based on CPNC bit. In case the CAN Partial Networking is enabled (CPNC is set), this function automatically sets the PNCOK bit as well.

## Parameters

in	<i>instance</i>	Index of SBC to configure
in	<i>cancConfig</i>	CAN control configuration object

**Returns**

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

**Note**

Part of the CAN control configuration is also the PNCOK setting. This bit confirms that CAN partial networking configuration is valid, however, it is cleared after a write access to any of the CAN partial networking configuration registers (0x26 - 0x2F, 0x68 - 0x6F).

To use the CAN partial networking but avoid unnecessary writes into the SBC the CAN partial networking configuration prior to [UJA113X\\_SetCan\(\)](#) call is recommended. Afterwards, this function can be called with CPNC set which automatically sets the PNCOK.

**Warning**

The PNCOK bit is cleared after a write access to any of the CAN partial networking configuration registers (0x26 - 0x2F, 0x68 - 0x6F)

**See also**

[UJA113X\\_SetCanPartNet\(\)](#) [UJA113X\\_SetDataMasks\(\)](#)

Definition at line 641 of file sbc\_uja113x.c.

**16.2.6.2** `status_t UJA113X_SetCanPartNet ( uint32_t instance, const uja113x_cpnc_cfg_t * cpncConfig )`

Sets the CAN partial networking control register (0x26 - 0x2F) based on input configuration object.

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the CAN partial networking control registers have not been written properly and the process has to be repeated.

**Parameters**

in	<i>instance</i>	Index of SBC to configure
in	<i>cpncConfig</i>	CAN partial networking configuration object

**Returns**

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

**Note**

It's possible to wake up on ID reception only and not evaluate the DLC & data content. From this perspective it would make no sense to configure data masks. Thus the data masks have to be configured with a separate function.

As the SPI protocol allows only 16/24/32 bit writes, four subsequent SPI transfers are carried out!

**See also**

[UJA113X\\_SetDataMasks\(\)](#)

Definition at line 746 of file sbc\_uja113x.c.

**16.2.6.3 status\_t UJA113X\_SetDataMasks ( uint32\_t instance, const uja113x\_dm\_cfg\_t \* dmConfig )**

Sets the CAN partial networking data mask registers (0x68 - 0x6F) based on input configuration object.

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the CAN partial networking data mask registers have not been written properly and the process has to be repeated.

**Parameters**

in	<i>instance</i>	Index of SBC to configure
in	<i>dmConfig</i>	CAN partial networking data masks configuration object

**Returns**

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

**Note**

This function configures all the data masks. In case the user is interested in configuring a single mask, a general function for SBC register write can be used instead!  
As the SPI protocol allows only 16/24/32 bit writes, three subsequent SPI transfers are carried out!

**See also**

UJA113X\_WriteRegister()

Definition at line 827 of file sbc\_uja113x.c.

## 16.3 01. Mode handling

### 16.3.1 Detailed Description

#### 16.3.2 Introduction

The SBC has two debug modes, the Forced Normal Mode (FNMC) and Software Development mode (SDMC). These modes can be configured in the SBCs MTPNV memory and can be set using the [UJA113X\\_SetMtpnv\(\)](#) function. Be aware while writing the non-volatile memory, as these registers can't be rewritten again easily. Clearing the non-volatile memory requires special treatment, see [Restoring MTPNV factory preset values](#) for details. These modes influence the watchdog behaviour and the main purpose is to use them during development, although is not limited just for this use case.

Additionally, the SBC has 3 operating modes the user can switch to (the Normal, Standby and Sleep mode) with [UJA113X\\_SetMode\(\)](#).

##### 16.3.2.1 Sleep mode

To be able to switch to the Sleep mode, the MTPNV memory has to be set accordingly. In case the Sleep mode is disabled, the request to switch to Sleep mode is ignored.

To enter the sleep mode, any pending interrupts or wake-up events need to be cleared and at least one regular wake-up source has to be enabled. Otherwise a system reset is triggered and the Reset Source Status bit (RSS) is set to 10100 (illegal Sleep mode command received).

##### 16.3.2.2 Normal & Standby mode

There should be no limitation to switch into these two modes except constraints related to watchdog. As the watchdog can be still running, make sure to handle the watchdog properly, otherwise a reset can occur.

#### Enumerations

- enum [uja113x\\_mc\\_mc\\_t](#) { [MC\\_SLEEP](#) = 0x01U, [MC\\_STANDBY](#) = 0x04U, [MC\\_NORMAL](#) = 0x07U }

*SBC mode configuration options (0x01:MC[2:0])*

#### Functions

- status\_t [UJA113X\\_SetMode](#) (uint32\_t instance, [uja113x\\_mc\\_mc\\_t](#) modeConfig)

*Sets the mode control register (0x01) to get the SBC into a new mode.*

### 16.3.3 Enumeration Type Documentation

#### 16.3.3.1 enum [uja113x\\_mc\\_mc\\_t](#)

SBC mode configuration options (0x01:MC[2:0])

Implements : [uja113x\\_mc\\_mc\\_t\\_Class](#)

#### Enumerator

**[MC\\_SLEEP](#)** Set SBC mode to Sleep.

**[MC\\_STANDBY](#)** Set SBC mode to Standby.

**[MC\\_NORMAL](#)** Set SBC mode to Normal.

Definition at line 257 of file [sbc\\_uja113x.h](#).

#### 16.3.4 Function Documentation

##### 16.3.4.1 `status_t UJA113X_SetMode ( uint32_t instance, uja113x_mc_mc_t modeConfig )`

Sets the mode control register (0x01) to get the SBC into a new mode.

The function prepares a frame based on input parameter and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the watchdog register has not been written and the process has to be repeated.

##### Parameters

in	<i>instance</i>	Index of SBC to configure
in	<i>modeConfig</i>	New mode

##### Returns

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

##### Note

The SBC can be set into three modes (Normal, Standby, Sleep) based on the input parameter.

##### Warning

Entering the Normal or Standby mode requires no special treatment. However entering a Sleep mode requires to clear all pending interrupts/events (which would set the SBCs INTN1/INTN2 to high) and at least one wake-up source to be enabled. Otherwise the SBC can get stuck at Sleep mode without a possibility to wake it up!

Definition at line 411 of file `sbc_uja113x.c`.

## 16.4 01. Primary control

### 16.4.1 Detailed Description

The primary control registers (0x00 - 0x1F) allow:

- Configuring watchdog (0x00)
- Changing SBC mode (0x01)
- Configuring Fail-safe behaviour (0x02)
- Read SBCs main status (0x03)
- Enable system interrupts (0x04)
- Read watchdog status (0x05)
- Access (read/write) general purpose memory 0 - 3 (0x06 - 0x09)
- Lock specific memory areas (0x0A)

Interrupts and interrupt status is described in [06. Interrupts](#).

#### Modules

- [01. Mode handling](#)
- [02. Watchdog](#)
- [03. Fail-safe configuration](#)
- [04. Lock control](#)



## 16.5 01. Regulator configuration

### 16.5.1 Detailed Description

#### 16.5.2 Introduction

The SBC incorporates two voltage regulators. Configurations influencing behaviour during start-up are stored in the MTPNV memory (see [Start-up control configuration options \(0x73\)](#) & [SBC configuration control options \(0x74\)](#)). After successful start-up, the regulators can be configured by invoking the [UJA113X\\_SetRegulator\(\)](#).

#### Data Structures

- struct [uja113x\\_rc\\_cfg\\_t](#)

*Regulator control register configuration structure (0x10) [More...](#)*

#### Enumerations

- enum [uja113x\\_rc\\_v2sc\\_t](#) { [V2SC\\_NONE](#) = 0x00U, [V2SC\\_UNDRVOLTAGE](#) = 0x10U, [V2SC\\_OVERVOLTAGE](#) = 0x20U, [V2SC\\_FULL](#) = 0x30U }

*V2 shut-down response to over- or undervoltage configuration options (0x10:V2SC[5:4])*

- enum [uja113x\\_rc\\_v2c\\_t](#) { [V2C\\_OFF](#) = 0x00U, [V2C\\_ON\\_N](#) = 0x04U, [V2C\\_ON\\_NSR](#) = 0x08U, [V2C\\_ON\\_NSSRF](#) = 0x0CU }

*V2 control configuration options (0x10:V2C[3:2])*

- enum [uja113x\\_rc\\_v1rtc\\_t](#) { [V1RTC\\_90\\_NOMINAL\\_OUT](#) = 0x00U, [V1RTC\\_80\\_NOMINAL\\_OUT](#) = 0x01U, [V1RTC\\_70\\_NOMINAL\\_OUT](#) = 0x02U, [V1RTC\\_60\\_NOMINAL\\_OUT](#) = 0x03U }

*V1 undervoltage reset threshold configuration options (0x10:V1RTC[1:0])*

#### Functions

- status\_t [UJA113X\\_SetRegulator](#) (uint32\_t instance, const [uja113x\\_rc\\_cfg\\_t](#) \*rcConfig)

*Sets the regulator control register (0x10) based on input configuration object.*

### 16.5.3 Data Structure Documentation

#### 16.5.3.1 struct [uja113x\\_rc\\_cfg\\_t](#)

Regulator control register configuration structure (0x10)

Implements : [uja113x\\_rc\\_cfg\\_t\\_Class](#)

Definition at line 1451 of file [sbc\\_uja113x.h](#).

#### Data Fields

- [uja113x\\_rc\\_v2sc\\_t](#) v2ShutdownCtrl

*Configures V2 shut-down response to a battery over- or undervoltage configuration (0x10:V2SC[5:4])*

- [uja113x\\_rc\\_v2c\\_t](#) v2Ctrl

*Configures output behavior on V2 (0x10:V2C[3:2])*

- [uja113x\\_rc\\_v1rtc\\_t](#) v1RstThrCtrl

*Configures V1 undervoltage reset threshold (0x10:V1RTC[1:0])*

## Field Documentation

16.5.3.1.1 `uja113x_rc_v1rtc_t v1RstThrCtrl`

Configures V1 undervoltage reset threshold (0x10:V1RTC[1:0])

Definition at line 1464 of file `sbc_uja113x.h`.

16.5.3.1.2 `uja113x_rc_v2c_t v2Ctrl`

Configures output behavior on V2 (0x10:V2C[3:2])

Definition at line 1460 of file `sbc_uja113x.h`.

16.5.3.1.3 `uja113x_rc_v2sc_t v2ShutdownCtrl`

Configures V2 shut-down response to a battery over- or undervoltage configuration (0x10:V2SC[5:4])

Definition at line 1456 of file `sbc_uja113x.h`.

## 16.5.4 Enumeration Type Documentation

16.5.4.1 `enum uja113x_rc_v1rtc_t`

V1 undervoltage reset threshold configuration options (0x10:V1RTC[1:0])

Implements : `uja113x_rc_v1rtc_t_Class`

## Enumerator

**`V1RTC_90_NOMINAL_OUT`** Reset threshold set to 90% of V1 nominal output voltage.

**`V1RTC_80_NOMINAL_OUT`** Reset threshold set to 80% of V1 nominal output voltage.

**`V1RTC_70_NOMINAL_OUT`** Reset threshold set to 70% of V1 nominal output voltage.

**`V1RTC_60_NOMINAL_OUT`** Reset threshold set to 60% of V1 nominal output voltage.

Definition at line 403 of file `sbc_uja113x.h`.

16.5.4.2 `enum uja113x_rc_v2c_t`

V2 control configuration options (0x10:V2C[3:2])

Implements : `uja113x_rc_v2c_t_Class`

## Enumerator

**`V2C_OFF`** V2 off in all modes.

**`V2C_ON_N`** V2 on in Normal mode.

**`V2C_ON_NSR`** V2 on in Normal, Standby and Reset modes.

**`V2C_ON_NSSRF`** V2 on in Normal, Standby, Sleep, Reset and FSP mode.

Definition at line 388 of file `sbc_uja113x.h`.

16.5.4.3 `enum uja113x_rc_v2sc_t`

V2 shut-down response to over- or undervoltage configuration options (0x10:V2SC[5:4])

Implements : `uja113x_rc_v2sc_t_Class`

## Enumerator

**`V2SC_NONE`** No shut-down in response to over- or undervoltage.

**`V2SC_UNDRVOLTAGE`** Shut-down in response to undervoltage.

**V2SC\_OVERVOLTAGE** Shut-down in response to overvoltage.

**V2SC\_FULL** Shut-down in response to over- or undervoltage.

Definition at line 373 of file sbc\_uja113x.h.

#### 16.5.5 Function Documentation

##### 16.5.5.1 status\_t UJA113X\_SetRegulator ( uint32\_t instance, const uja113x\_rc\_cfg\_t \* rcConfig )

Sets the regulator control register (0x10) based on input configuration object.

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the regulator control register has not been written and the process has to be repeated.

##### Parameters

in	<i>instance</i>	Index of SBC to configure
in	<i>rcConfig</i>	Regulator control configuration object

##### Returns

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

Definition at line 519 of file sbc\_uja113x.c.

## 16.6 02. Battery monitor configuration

### 16.6.1 Detailed Description

### 16.6.2 Introduction

The battery monitor can measure the voltage either on BAT or BATSENSE pin and is able to generate an interrupt in case of under-/overvoltage in Normal mode.

To read the content of the ADC, the [UJA113X\\_SbcTransfer\(\)](#) function can be used. No dedicated function for reading the ADC is available at the moment.

[UJA113X\\_SetBatteryMonitor\(\)](#) can be used to configure the whole battery monitor except interrupts.

### Data Structures

- struct [uja113x\\_bmc\\_cfg\\_t](#)

*Battery monitor registers configuration structure (0x11 - 0x14) [More...](#)*

### Typedefs

- typedef bool [uja113x\\_bmetsc\\_bmsc\\_t](#)  
*Battery monitor event trigger source control (0x11:BMSC[0])*
- typedef uint8\_t [uja113x\\_bmutc\\_bmutc\\_t](#)  
*Battery monitor undervoltage threshold control parameter (0x12:BMUTC[7:0])*
- typedef uint8\_t [uja113x\\_bmotc\\_bmotc\\_t](#)  
*Battery monitor overvoltage threshold control parameter (13:BMOTC[7:0])*
- typedef uint8\_t [uja113x\\_bmhc\\_bmhc\\_t](#)  
*Battery monitor hysteresis for overvoltage control parameter (0x14:BMHOC[7:4])*
- typedef uint8\_t [uja113x\\_bmhc\\_bmhc\\_t](#)  
*Battery monitor hysteresis for undervoltage control parameter (0x14:BMHUC[3:0])*

### Enumerations

- enum [uja113x\\_bscdt\\_ionsctc\\_t](#) { [IONSCTC\\_30\\_MA](#) = 0U, [IONSCTC\\_45\\_MA](#) = 1U, [IONSCTC\\_60\\_MA](#) = 2U, [IONSCTC\\_90\\_MA](#) = 3U }  
*Bank N short-circuit detection threshold configuration options (0x39 / 0x49)*
- enum [uja113x\\_boldt\\_ionoltc\\_t](#) { [IONOLTC\\_2\\_MA](#) = 0U, [IONOLTC\\_5\\_MA](#) = 1U, [IONOLTC\\_10\\_MA](#) = 2U, [IONOLTC\\_20\\_MA](#) = 3U }  
*Bank N open-load detection threshold configuration options (0x3A / 0x4A)*

### Functions

- status\_t [UJA113X\\_SetBatteryMonitor](#) (uint32\_t instance, const [uja113x\\_bmc\\_cfg\\_t](#) \*bmcConfig)  
*Sets the Battery monitor control registers (0x11 - 0x14) based on input configuration object.*

### 16.6.3 Data Structure Documentation

#### 16.6.3.1 struct [uja113x\\_bmc\\_cfg\\_t](#)

Battery monitor registers configuration structure (0x11 - 0x14)

Implements : [uja113x\\_bmc\\_cfg\\_t\\_Class](#)

Definition at line 1474 of file [sbc\\_uja113x.h](#).

## Data Fields

- [uja113x\\_bmetsc\\_bmsc\\_t bmSrcCtrl](#)  
*Selects the battery monitor event trigger source (0x11:BMSC[0])*
- [uja113x\\_bmutc\\_bmutc\\_t bmUndrVoltThr](#)  
*Configures the battery monitor undervoltage threshold (0x12:BMUTC[7:0])*
- [uja113x\\_bmotc\\_bmotc\\_t bmOverVoltThr](#)  
*Configures the battery monitor overvoltage threshold (0x13:BMOTC[7:0])*
- [uja113x\\_bmhc\\_bmhc\\_t bmOverVoltThrRel](#)  
*Configures the battery monitor hysteresis for overvoltage (0x14:BMHOC[7:4])*
- [uja113x\\_bmhc\\_bmhc\\_t bmUndrVoltThrRel](#)  
*Configures the battery monitor hysteresis for undervoltage (0x14:BMHUC[3:0])*

## Field Documentation

16.6.3.1.1 [uja113x\\_bmotc\\_bmotc\\_t bmOverVoltThr](#)

Configures the battery monitor overvoltage threshold (0x13:BMOTC[7:0])

Threshold for triggering a battery overvoltage event and BMOI interrupt

**threshold** =  $\text{BMOTC}[7:0]/255 \times 20\text{V}$

Allowed range: 0 - 255

Definition at line 1491 of file sbc\_uja113x.h.

16.6.3.1.2 [uja113x\\_bmhc\\_bmhc\\_t bmOverVoltThrRel](#)

Configures the battery monitor hysteresis for overvoltage (0x14:BMHOC[7:4])

Battery monitor overvoltage threshold release level;

**release level** =  $\text{BMHOC}[7:4] \times 4/255 \times 20\text{V}$  below threshold defined by BMOTC

## Warning

Only 4 lsb are used! The upper part of byte is truncated!

Definition at line 1498 of file sbc\_uja113x.h.

16.6.3.1.3 [uja113x\\_bmetsc\\_bmsc\\_t bmSrcCtrl](#)

Selects the battery monitor event trigger source (0x11:BMSC[0])

Definition at line 1479 of file sbc\_uja113x.h.

16.6.3.1.4 [uja113x\\_bmutc\\_bmutc\\_t bmUndrVoltThr](#)

Configures the battery monitor undervoltage threshold (0x12:BMUTC[7:0])

Threshold for triggering a battery undervoltage event and BMUI interrupt;

**threshold** =  $\text{BMUTC}[7:0]/255 \times 20\text{V}$ .

Allowed range: 0 - 255

Definition at line 1485 of file sbc\_uja113x.h.

16.6.3.1.5 [uja113x\\_bmhc\\_bmhc\\_t bmUndrVoltThrRel](#)

Configures the battery monitor hysteresis for undervoltage (0x14:BMHUC[3:0])

Battery monitor undervoltage threshold release level;

**release level** =  $\text{BMHUC}[3:0] \times 4/255 \times 20\text{V}$  above threshold defined by BMUTC

**Warning**

Only 4 lsb are used! The upper part of byte is truncated!

Definition at line 1505 of file sbc\_uja113x.h.

**16.6.4 Typedef Documentation****16.6.4.1 typedef bool uja113x\_bmetsc\_bmsc\_t**

Battery monitor event trigger source control (0x11:BMSC[0])

false - BAT pin is source of event triggers

true - BATSENSE pin is source of event triggers

Implements : uja113x\_bmetsc\_bmsc\_t\_Class

Definition at line 420 of file sbc\_uja113x.h.

**16.6.4.2 typedef uint8\_t uja113x\_bmhc\_bmhoc\_t**

Battery monitor hysteresis for overvoltage control parameter (0x14:BMHOC[7:4])

Implements uja113x\_bmhc\_bmhoc\_t\_Class

Definition at line 447 of file sbc\_uja113x.h.

**16.6.4.3 typedef uint8\_t uja113x\_bmhc\_bmhuc\_t**

Battery monitor hysteresis for undervoltage control parameter (0x14:BMHUC[3:0])

Implements uja113x\_bmhc\_bmhuc\_t\_Class

Definition at line 456 of file sbc\_uja113x.h.

**16.6.4.4 typedef uint8\_t uja113x\_bmotc\_bmotc\_t**

Battery monitor overvoltage threshold control parameter (13:BMOTC[7:0])

Implements : uja113x\_bmotc\_bmotc\_t\_Class Allowed range: 0 - 255

Definition at line 438 of file sbc\_uja113x.h.

**16.6.4.5 typedef uint8\_t uja113x\_bmutc\_bmutc\_t**

Battery monitor undervoltage threshold control parameter (0x12:BMUTC[7:0])

Implements : uja113x\_bmutc\_bmutc\_t\_Class Allowed range: 0 - 255

Definition at line 429 of file sbc\_uja113x.h.

**16.6.5 Enumeration Type Documentation****16.6.5.1 enum uja113x\_boldt\_ionoltc\_t**

Bank N open-load detection threshold configuration options (0x3A / 0x4A)

**Note**

For details, consult the reference manual - "Static characteristics: lth(det)open"

Implements : uja113x\_boldt\_ionoltc\_t\_Class

**Enumerator**

**IONOLTC\_2\_MA** HVIO N open-load detection threshold set to 2 mA.

**IONOLTC\_5\_MA** HVIO N open-load detection threshold set to 5 mA.

**IONOLTC\_10\_MA** HVIO N open-load detection threshold set to 10 mA.

**IONOLTC\_20\_MA** HVIO N open-load detection threshold set to 20 mA.

Definition at line 1065 of file sbc\_uja113x.h.

#### 16.6.5.2 enum uja113x\_bscdt\_ionsctc\_t

Bank N short-circuit detection threshold configuration options (0x39 / 0x49)

##### Note

For details consult the reference manual - "Static characteristics: lth(det)sc"

Implements : uja113x\_bscdt\_ionsctc\_t\_Class

##### Enumerator

**IONSCTC\_30\_MA** HVIO N short-circuit detection threshold set to 30 mA.

**IONSCTC\_45\_MA** HVIO N short-circuit detection threshold set to 45 mA.

**IONSCTC\_60\_MA** HVIO N short-circuit detection threshold set to 60 mA.

**IONSCTC\_90\_MA** HVIO N short-circuit detection threshold set to 90 mA.

Definition at line 1050 of file sbc\_uja113x.h.

#### 16.6.6 Function Documentation

##### 16.6.6.1 status\_t UJA113X\_SetBatteryMonitor ( uint32\_t instance, const uja113x\_bmc\_cfg\_t \* bmcConfig )

Sets the Battery monitor control registers (0x11 - 0x14) based on input configuration object.

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the regulator control registers were not written properly and the process has to be repeated.

##### Parameters

in	<i>instance</i>	Index of SBC to configure
in	<i>bmcConfig</i>	Battery monitor control configuration object

##### Returns

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

##### Note

The function configures all the battery monitor features:

1. Battery monitor event trigger source (0x11)
2. Battery monitor undervoltage threshold (0x12)
3. Battery monitor overvoltage threshold (0x13)
4. Battery monitor hysteresis (0x14)

As the SPI protocol allows only 16/24/32 bit writes, two subsequent SPI transfers are carried out!

Definition at line 546 of file sbc\_uja113x.c.

## 16.7 02. LIN configuration

### 16.7.1 Detailed Description

### 16.7.2 Introduction

A UJA113x device can incorporate up to 2 LIN transceivers minimum of 1. Configurable LIN features are:

- LIN slope
- LIN mode

To configure the LIN transceiver, [UJA113X\\_SetLin\(\)](#) function can be used.

#### Warning

In case the user decides to implement his own function or use the [UJA113X\\_SbcTransfer\(\)](#), to write the LIN control register (0x21), make sure LIN2 bits are cleared for devices with a single LIN transceiver (see reference manual, register 0x21 description, for details).

#### Data Structures

- struct [uja113x\\_linc\\_cfg\\_t](#)  
*LIN control register configuration structure (0x21) [More...](#)*

#### Enumerations

- enum [uja113x\\_linc\\_lsc2\\_t](#) { [LSC2\\_SLOPE\\_ACTIVE](#) = 0x00U, [LSC2\\_SLOPE\\_ACTIVE2](#) = 0x40U, [LSC2\\_SLOPE\\_ACTIVE\\_TXDL\\_TIMEOUT](#) = 0x80U }  
*LIN2 slope control configuration options (0x21:LSC2[7:6])*
- enum [uja113x\\_linc\\_lmc2\\_t](#) { [LMC2\\_TRANSCEIVER\\_OFF](#) = 0x00U, [LMC2\\_TRANSCEIVER\\_ON](#) = 0x10U, [LMC2\\_TRANSCEIVER\\_ON2](#) = 0x20U, [LMC2\\_TRANSCEIVER\\_LISTEN](#) = 0x30U }  
*LIN2 transceiver operating mode configuration options (0x21:LMC2[5:4])*
- enum [uja113x\\_linc\\_lsc1\\_t](#) { [LSC1\\_SLOPE\\_ACTIVE](#) = 0x00U, [LSC1\\_SLOPE\\_ACTIVE2](#) = 0x04U, [LSC1\\_SLOPE\\_ACTIVE\\_TXDL\\_TIMEOUT](#) = 0x08U }  
*LIN/LIN1 slope control configuration options (0x21:LSC1[3:2])*
- enum [uja113x\\_linc\\_lmc1\\_t](#) { [LMC1\\_TRANSCEIVER\\_OFF](#) = 0x00U, [LMC1\\_TRANSCEIVER\\_ON](#) = 0x01U, [LMC1\\_TRANSCEIVER\\_ON2](#) = 0x02U, [LMC1\\_TRANSCEIVER\\_LISTEN](#) = 0x03U }  
*LIN/LIN1 transceiver operating mode configuration options (0x21:LMC1[1:0])*

#### Functions

- status\_t [UJA113X\\_SetLin](#) (uint32\_t instance, const [uja113x\\_linc\\_cfg\\_t](#) \*lincConfig)  
*Sets the LIN control register (0x21) based on input configuration object.*

### 16.7.3 Data Structure Documentation

#### 16.7.3.1 struct [uja113x\\_linc\\_cfg\\_t](#)

LIN control register configuration structure (0x21)

Implements : [uja113x\\_linc\\_cfg\\_t\\_Class](#)

Definition at line 1565 of file [sbc\\_uja113x.h](#).



## Data Fields

- [uja113x\\_linc\\_lsc2\\_t lin2SlopeCtrl](#)  
*Configures LIN2 slope control behaviour (0x21:LSC2[7:6])*
- [uja113x\\_linc\\_lmc2\\_t lin2ModeCtrl](#)  
*Configures LIN2 transceiver operating mode (0x21:LMC2[5:4])*
- [uja113x\\_linc\\_lsc1\\_t lin1SlopeCtrl](#)  
*Configures LIN1 slope control behaviour (0x21:LSC1[3:2])*
- [uja113x\\_linc\\_lmc1\\_t lin1ModeCtrl](#)  
*Configures LIN1 transceiver operating mode (0x21:LMC1[1:0])*

## Field Documentation

16.7.3.1.1 [uja113x\\_linc\\_lmc1\\_t lin1ModeCtrl](#)

Configures LIN1 transceiver operating mode (0x21:LMC1[1:0])

Definition at line 1582 of file sbc\_uja113x.h.

16.7.3.1.2 [uja113x\\_linc\\_lsc1\\_t lin1SlopeCtrl](#)

Configures LIN1 slope control behaviour (0x21:LSC1[3:2])

Definition at line 1578 of file sbc\_uja113x.h.

16.7.3.1.3 [uja113x\\_linc\\_lmc2\\_t lin2ModeCtrl](#)

Configures LIN2 transceiver operating mode (0x21:LMC2[5:4])

Definition at line 1574 of file sbc\_uja113x.h.

16.7.3.1.4 [uja113x\\_linc\\_lsc2\\_t lin2SlopeCtrl](#)

Configures LIN2 slope control behaviour (0x21:LSC2[7:6])

Definition at line 1570 of file sbc\_uja113x.h.

## 16.7.4 Enumeration Type Documentation

16.7.4.1 [enum uja113x\\_linc\\_lmc1\\_t](#)

LIN/LIN1 transceiver operating mode configuration options (0x21:LMC1[1:0])

Implements : [uja113x\\_linc\\_lmc1\\_t\\_Class](#)

## Enumerator

- LMC1\_TRANSCEIVER\_OFF** Offline.
- LMC1\_TRANSCEIVER\_ON** Active mode (when SBC in normal mode).
- LMC1\_TRANSCEIVER\_ON2** Active mode (when SBC in normal mode).
- LMC1\_TRANSCEIVER\_LISTEN** Listen-only mode.

Definition at line 691 of file sbc\_uja113x.h.

16.7.4.2 [enum uja113x\\_linc\\_lmc2\\_t](#)

LIN2 transceiver operating mode configuration options (0x21:LMC2[5:4])

Implements : [uja113x\\_linc\\_lmc2\\_t\\_Class](#)

## Enumerator

**LMC2\_TRANSCEIVER\_OFF** Offline.

**LMC2\_TRANSCEIVER\_ON** Active mode (when SBC in normal mode).

**LMC2\_TRANSCEIVER\_ON2** Active mode (when SBC in normal mode).

**LMC2\_TRANSCEIVER\_LISTEN** Listen-only mode.

Definition at line 661 of file sbc\_uja113x.h.

## 16.7.4.3 enum uja113x\_linc\_lsc1\_t

LIN/LIN1 slope control configuration options (0x21:LSC1[3:2])

Implements : uja113x\_linc\_lsc1\_t\_Class

## Enumerator

**LSC1\_SLOPE\_ACTIVE** Slope control active.

**LSC1\_SLOPE\_ACTIVE2** Slope control active.

**LSC1\_SLOPE\_ACTIVE\_TXDL\_TIMEOUT** Slope control active and TXDL dominant time-out deactivated.

Definition at line 676 of file sbc\_uja113x.h.

## 16.7.4.4 enum uja113x\_linc\_lsc2\_t

LIN2 slope control configuration options (0x21:LSC2[7:6])

Implements : uja113x\_linc\_lsc2\_t\_Class

## Enumerator

**LSC2\_SLOPE\_ACTIVE** Slope control active.

**LSC2\_SLOPE\_ACTIVE2** Slope control active.

**LSC2\_SLOPE\_ACTIVE\_TXDL\_TIMEOUT** Slope control active and TXDL dominant time-out deactivated.

Definition at line 646 of file sbc\_uja113x.h.

## 16.7.5 Function Documentation

## 16.7.5.1 status\_t UJA113X\_SetLin ( uint32\_t instance, const uja113x\_linc\_cfg\_t \* lincConfig )

Sets the LIN control register (0x21) based on input configuration object.

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the LIN control register has not been written properly and the process has to be repeated.

## Parameters

in	instance	Index of SBC to configure
in	lincConfig	LIN control configuration object

## Returns

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

**Note**

The reference manual states, that bits 7:4 (LIN2 configuration) of register 0x21 should remain cleared for devices incorporating only LIN1. Therefore for these devices the function clears bits even when configured to non-zero value!

Definition at line 675 of file sbc\_uja113x.c.

## 16.8 02. Supply control

### 16.8.1 Detailed Description

The supply control registers (0x10 - 0x1F) allow:

- Configuring voltage regulators (0x10)
- Configuring battery monitor (0x11 - 0x18)
- Configuring switched Mode Power Supply (0x19 - 0x1A)
- Read supply voltage status (0x1B)
- Enable supply interrupts (0x1C)

Interrupts and interrupt status is described in [06. Interrupts](#).

#### Modules

- [01. Regulator configuration](#)
- [02. Battery monitor configuration](#)
- [03. SMPS configuration](#)

## 16.9 02. Watchdog

### 16.9.1 Detailed Description

#### 16.9.2 Introduction

The watchdog behaviour can be influenced by the debug modes (see mode [Introduction](#) for details) in combination with the watchdog mode (window, timeout or autonomous mode). The watchdog period is adjustable ranging from 8 ms to 4094 ms.

##### 16.9.2.1 Watchdog in FNMC

In Forced Normal Mode the Watchdog is completely disabled. Thus, configuring the watchdog mode has no influence on it's operation!

##### 16.9.2.2 Watchdog in SDMC

In Software Development Mode the watchdog can be activated or disabled based on configured watchdog mode. In case the timeout or window mode is selected, the watchdog is active independent from SDMC. For autonomous mode, the watchdog is active (only timeout mode) while SDMC = 0. For SDMC = 1, the watchdog stays inactive in autonomous mode!

Following table summarizes all the use cases.

SBC state	Watchdog Configurations			
	SDMC = x WMC = 100 (Window)	SDMC = x WMC = 010 (Timeout)	SDMC = 0 WMC = 001 (Autonomous)	SDMC = 1 WMC = 001 (Autonomous)
Normal mode	Window	Timeout	Timeout	Off
Standby mode (INTN1 HIGH)	Timeout	Timeout	Off	Off
Standby mode (INTN1 LOW)	Timeout	Timeout	Timeout	Off
Sleep mode	Timeout	Timeout	Off	Off
Forced Normal mode	Off	Off	Off	Off
Other modes	Off	Off	Off	Off

Table 1: Summary of Watchdog settings

##### 16.9.2.3 Watchdog mode configuration

The watchdog mode can be reconfigured only in Standby mode using the [UJA113X\\_SetWatchdog\(\)](#). Writes leading to change the watchdog mode in other modes than Standby would cause a switch to reset mode and reset source status (RSS) bit will be set to 10000 (illegal watchdog mode control access).

From this perspective it's the users responsibility to switch the SBC into Standby mode prior to watchdog mode update.

##### 16.9.2.4 Watchdog trigger

To reset the watchdog timer a successful write into the watchdog control register has to be performed. The [UJA113X\\_SetWatchdog\(\)](#) function can be used. If a new mode/period have changed as a result of the write access, the new values are valid immediately.

#### Warning

Writing a different mode while resetting the watchdog timer can cause a reset in case the SBC is not in Standby mode.

Writing a different period while resetting the watchdog timer can result in reset as a consequence of a watchdog counter overflow due to period change.

## Data Structures

- struct [uja113x\\_wtdc\\_cfg\\_t](#)  
Watchdog register configuration structure (reg 0x00) [More...](#)

## Enumerations

- enum [uja113x\\_wtdc\\_wmc\\_t](#) { [WMC\\_AUTONOMOUS](#) = 0x20U, [WMC\\_TIMEOUT](#) = 0x40U, [WMC\\_WINDOW](#) = 0x80U }  
Watchdog mode configuration options (0x00:WMC[7:5])
- enum [uja113x\\_wtdc\\_nwp\\_t](#) {  
[NWP\\_PERIOD\\_8](#) = 0x08U, [NWP\\_PERIOD\\_16](#) = 0x01U, [NWP\\_PERIOD\\_32](#) = 0x02U, [NWP\\_PERIOD\\_64](#) = 0x0BU,  
[NWP\\_PERIOD\\_128](#) = 0x04U, [NWP\\_PERIOD\\_256](#) = 0x0DU, [NWP\\_PERIOD\\_1024](#) = 0x0EU, [NWP\\_PERIOD\\_4096](#) = 0x07U }  
Watchdog nominal period configuration options (0x00:NWP[3:0])

## Functions

- status\_t [UJA113X\\_SetWatchdog](#) (uint32\_t instance, const [uja113x\\_wtdc\\_cfg\\_t](#) \*wtdcConfig)  
Sets the watchdog register (0x00) based on input configuration object.

### 16.9.3 Data Structure Documentation

#### 16.9.3.1 struct [uja113x\\_wtdc\\_cfg\\_t](#)

Watchdog register configuration structure (reg 0x00)

Implements : [uja113x\\_wtdc\\_cfg\\_t\\_Class](#)

Definition at line 1373 of file [sbc\\_uja113x.h](#).

#### Data Fields

- [uja113x\\_wtdc\\_wmc\\_t wtdModeCtrl](#)  
Configures the Watchdog Mode (0x00:WMC[7:5])
- [uja113x\\_wtdc\\_nwp\\_t nomWtdPeriod](#)  
Configures the watchdog nominal period (0x00:NWP[3:0])

#### Field Documentation

##### 16.9.3.1.1 [uja113x\\_wtdc\\_nwp\\_t nomWtdPeriod](#)

Configures the watchdog nominal period (0x00:NWP[3:0])

Definition at line 1382 of file [sbc\\_uja113x.h](#).

##### 16.9.3.1.2 [uja113x\\_wtdc\\_wmc\\_t wtdModeCtrl](#)

Configures the Watchdog Mode (0x00:WMC[7:5])

Definition at line 1378 of file [sbc\\_uja113x.h](#).

### 16.9.4 Enumeration Type Documentation

#### 16.9.4.1 enum [uja113x\\_wtdc\\_nwp\\_t](#)

Watchdog nominal period configuration options (0x00:NWP[3:0])

Implements : `uja113x_wtdc_nwp_t_Class`

#### Enumerator

***NWP\_PERIOD\_8*** 8 ms nominal period.  
***NWP\_PERIOD\_16*** 16 ms nominal watchdog period.  
***NWP\_PERIOD\_32*** 32 ms nominal watchdog period.  
***NWP\_PERIOD\_64*** 64 ms nominal watchdog period.  
***NWP\_PERIOD\_128*** 128 ms nominal watchdog period.  
***NWP\_PERIOD\_256*** 256 ms nominal watchdog period.  
***NWP\_PERIOD\_1024*** 1024 ms nominal watchdog period.  
***NWP\_PERIOD\_4096*** 4096 ms nominal watchdog period.

Definition at line 238 of file `sbc_uja113x.h`.

#### 16.9.4.2 enum `uja113x_wtdc_wmc_t`

Watchdog mode configuration options (0x00:WMC[7:5])

Implements : `uja113x_wtdc_wmc_t_Class`

#### Enumerator

***WMC\_AUTONOMOUS*** Watchdog autonomous mode  
***WMC\_TIMEOUT*** Watchdog timeout mode.  
***WMC\_WINDOW*** Watchdog window mode.

Definition at line 224 of file `sbc_uja113x.h`.

### 16.9.5 Function Documentation

#### 16.9.5.1 `status_t UJA113X_SetWatchdog ( uint32_t instance, const uja113x_wtdc_cfg_t * wtdcConfig )`

Sets the watchdog register (0x00) based on input configuration object.

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the watchdog register has not been written and the process has to be repeated.

#### Parameters

<code>in</code>	<i>instance</i>	Index of SBC to configure
<code>in</code>	<i>wtdcConfig</i>	Watchdog configuration object

#### Returns

**STATUS\_SUCCESS** Configuration successful  
**STATUS\_BUSY** SPI peripheral is busy  
**STATUS\_TIMEOUT** SPI communication timed out, configuration aborted  
**SBC\_CMD\_ERROR** First received byte doesn't match the repeated address, configuration failed.

#### Warning

The watchdog mode can be re-configured in Standby mode only! Changing the watchdog mode in Normal mode will cause the UJA113x to switch to Reset mode and the reset source status bits (RSS) will be set to 10000 ( illegal watchdog mode control access). However writing the same values doesn't lead to SBC reset, but a watchdog trigger which clears the watchdog counter!

**Note**

To reset the watchdog counter, a successful write into the watchdog control register (0x00) has to be performed. This function can be used to reset the counter, however keep in mind that updating mode in non-↔ Standby modes leads to SBC reset!

Although updating the watchdog period is allowed, SBC reset due to an overflow can happen after configuring a shorter period!

**See also**

[UJA113X\\_SetMode\(\)](#)

Definition at line 385 of file sbc\_uja113x.c.



## 16.10 03. Fail-safe configuration

### 16.10.1 Detailed Description

#### 16.10.2 Introduction

The SBC incorporates following fail-safe features:

- EN pin
  - can control an external hardware
  - can operate as general purpose output when system is running properly
- LIMP pin
  - enables "limp home" hardware in case of ECU failure
- Advanced LIMP home function via HVIO 4 - 2 configurable in non-volatile memory (see MTPNV [Introduction](#) for details)
- Register locking (see lock control [Introduction](#) for details)
- Reset counter

##### 16.10.2.1 Reset mechanism

If the system resets, the SBC initiates a reset process, during which a reset counter is incremented. In case the counter equals 3 when the UJA113x enters reset mode, the SBC assumes that a serious failure has occurred and switches to Forced Sleep mode (FSP), enabling the limp home function. A properly designed system should reset this counter to avoid its overflow.

The [UJA113X\\_SetFailSafe\(\)](#) function which configures the fail-safe functionality can be also used to update the reset counter. However the parameters for EN pin and LIMP home pin should reflect the current configuration!

The reset counter can be (pre)configured to any value providing additional level of safety. For instance, setting the counter to 3 would mean no errors allowed (in case an error occurs, the system would be forced to FSP immediately)!

#### Data Structures

- struct [uja113x\\_fsc\\_cfg\\_t](#)  
Fail-safe register configuration structure (reg 0x02) [More...](#)

#### Typedefs

- typedef bool [uja113x\\_fsc\\_endc\\_t](#)  
EN high-side driver activation options (0x02:ENDC[5])
- typedef bool [uja113x\\_fsc\\_lhc\\_t](#)  
LIMP output configuration options (0x02:LHC[2])
- typedef uint8\_t [uja113x\\_fsc\\_rcc\\_t](#)  
SBC reset counter configuration option (0x02:RCC[1:0])

#### Enumerations

- enum [uja113x\\_fsc\\_ensc\\_t](#) { [ENSC\\_NONE](#) = 0x00U, [ENSC\\_UNDRVOLTAGE](#) = 0x40U, [ENSC\\_OVERVOLTAGE](#) = 0x80U, [ENSC\\_FULL](#) = 0xC0U }  
EN pin shut-down control configuration options (0x02:ENSC[7:6])
- enum [uja113x\\_fsc\\_enc\\_t](#) { [ENC\\_LOW](#) = 0x00U, [ENC\\_HIGH\\_NRS](#) = 0x08U, [ENC\\_HIGH\\_N](#) = 0x10U, [ENC\\_TIMER](#) = 0x18U }  
EN pin output control configuration options (0x02:ENC[4:3])

## Functions

- [status\\_t UJA113X\\_SetFailSafe](#) (uint32\_t instance, const [uja113x\\_fsc\\_cfg\\_t](#) \*fscConfig)

*Sets the fail-safe register (0x02) based on input configuration object.*

### 16.10.3 Data Structure Documentation

#### 16.10.3.1 struct [uja113x\\_fsc\\_cfg\\_t](#)

Fail-safe register configuration structure (reg 0x02)

Implements : [uja113x\\_fsc\\_cfg\\_t\\_Class](#)

Definition at line 1392 of file [sbc\\_uja113x.h](#).

#### Data Fields

- [uja113x\\_fsc\\_ensc\\_t enShutdownCtrl](#)  
*Configures the behaviour of EN pin during battery over-/undervoltage (0x02:ENSC[7:6])*
- [uja113x\\_fsc\\_endc\\_t enDriverCtrl](#)  
*Enables/disables the EN high-side driver (0x02:ENDC[5])*
- [uja113x\\_fsc\\_enc\\_t enCtrl](#)  
*Configures the output level behaviour of EN pin (0x02:ENC[4:3])*
- [uja113x\\_fsc\\_lhc\\_t limpHomeCtrl](#)  
*Configures the Limp pin output behaviour (0x02:LHC[2])*
- [uja113x\\_fsc\\_rcc\\_t resetCntCtrl](#)  
*Configures the Reset counter value (0x02:RCC[1:0])*

#### Field Documentation

##### 16.10.3.1.1 [uja113x\\_fsc\\_enc\\_t enCtrl](#)

Configures the output level behaviour of EN pin (0x02:ENC[4:3])

Definition at line 1405 of file [sbc\\_uja113x.h](#).

##### 16.10.3.1.2 [uja113x\\_fsc\\_endc\\_t enDriverCtrl](#)

Enables/disables the EN high-side driver (0x02:ENDC[5])

Definition at line 1401 of file [sbc\\_uja113x.h](#).

##### 16.10.3.1.3 [uja113x\\_fsc\\_ensc\\_t enShutdownCtrl](#)

Configures the behaviour of EN pin during battery over-/undervoltage (0x02:ENSC[7:6])

Definition at line 1397 of file [sbc\\_uja113x.h](#).

##### 16.10.3.1.4 [uja113x\\_fsc\\_lhc\\_t limpHomeCtrl](#)

Configures the Limp pin output behaviour (0x02:LHC[2])

Definition at line 1409 of file [sbc\\_uja113x.h](#).

##### 16.10.3.1.5 [uja113x\\_fsc\\_rcc\\_t resetCntCtrl](#)

Configures the Reset counter value (0x02:RCC[1:0])

Definition at line 1413 of file [sbc\\_uja113x.h](#).

## 16.10.4 Typedef Documentation

## 16.10.4.1 typedef bool uja113x\_fsc\_endc\_t

EN high-side driver activation options (0x02:ENDC[5])

false - EN high-side driver enabled; push-pull output

true - EN high-side driver disabled; open-drain low-side driver

Implements : uja113x\_fsc\_endc\_t\_Class

Definition at line 288 of file sbc\_uja113x.h.

## 16.10.4.2 typedef bool uja113x\_fsc\_lhc\_t

LIMP output configuration options (0x02:LHC[2])

false - LIMP pin is floating

true - LIMP pin is driven LOW

Implements : uja113x\_fsc\_lhc\_t\_Class

Definition at line 316 of file sbc\_uja113x.h.

## 16.10.4.3 typedef uint8\_t uja113x\_fsc\_rcc\_t

SBC reset counter configuration option (0x02:RCC[1:0])

Implements : uja113x\_fsc\_rcc\_t\_Class0 - 3

Definition at line 325 of file sbc\_uja113x.h.

## 16.10.5 Enumeration Type Documentation

## 16.10.5.1 enum uja113x\_fsc\_enc\_t

EN pin output control configuration options (0x02:ENC[4:3])

Implements : uja113x\_fsc\_enc\_t\_Class

## Enumerator

**ENC\_LOW** EN pin is driven permanently low.

**ENC\_HIGH\_NRS** EN pin is high (or floating if ENDC = 1) when SBC is in Normal, Reset or Standby mode.

**ENC\_HIGH\_N** EN pin is high (or floating if ENDC = 1) when SBC is in Normal mode.

**ENC\_TIMER** EN pin is controller by Timer 2.

Definition at line 297 of file sbc\_uja113x.h.

## 16.10.5.2 enum uja113x\_fsc\_ensc\_t

EN pin shut-down control configuration options (0x02:ENSC[7:6])

Implements : uja113x\_fsc\_ensc\_t\_Class

## Enumerator

**ENSC\_NONE** EN pin not influenced by battery over- or undervoltage.

**ENSC\_UNDRVOLTAGE** EN pin driven LOW when battery undervoltage detected.

**ENSC\_OVERVOLTAGE** EN pin driven LOW when battery overvoltage detected.

**ENSC\_FULL** EN pin driven LOW when battery over- or undervoltage detected.

Definition at line 271 of file sbc\_uja113x.h.

### 16.10.6 Function Documentation

#### 16.10.6.1 `status_t UJA113X_SetFailSafe ( uint32_t instance, const uja113x_fsc_cfg_t * fscConfig )`

Sets the fail-safe register (0x02) based on input configuration object.

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the fail-safe control register has not been written and the process has to be repeated.

##### Parameters

in	<i>instance</i>	Index of SBC to configure
in	<i>fscConfig</i>	Fail-safe configuration object

##### Returns

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

##### Warning

Care has to be taken while configuring the reset counter (resetCntCtrl). In case the SBC reset counter is set to 3 and a reset occurs, the SBC will be switched into Forced Sleep Mode! This can be used as a safety measure, which doesn't allow any SBC resets, however can lead to unexpected behavior if configured improperly! Read the datasheet carefully to configure this parameter the right way.

##### Note

After every SBC reset the reset counter is incremented. To reset the counter, call this function with a requested reset counter value.

Bear in mind that "clearing" the reset counter requires you to write the whole register. This can clear your preconfigured values from SBC initialization step leading to unwanted behavior. So in case of reset counter reset, the rest of parameters has to be configured as well!

Definition at line 435 of file sbc\_uja113x.c.

## 16.11 03. SMPS configuration

### 16.11.1 Detailed Description

#### 16.11.2 Introduction

The Switched Mode Power Supply (SMPS) can be configured by invoking the [UJA113X\\_SetSmpls\(\)](#) function.

#### Data Structures

- struct [uja113x\\_smpsc\\_cfg\\_t](#)  
*SMPS (buck-boost converter) registers configuration structure (0x19 - 0x1A) [More...](#)*

#### Typedefs

- typedef bool [uja113x\\_smpsc\\_smpsotc\\_t](#)  
*SMPS overtemperature configuration options (0x19:SMPSOTC[3])*

#### Enumerations

- enum [uja113x\\_smpsc\\_smpsc\\_t](#) { [SMPSC\\_ON\\_NSR](#) = 0x00U, [SMPSC\\_ON\\_NSRS](#) = 0x01U, [SMPSC\\_PASST\\_THROUGH\\_NSS](#) = 0x03U }  
*SMPS on/off configuration options (0x19:SMPSOC[1:0])*
- enum [uja113x\\_smpsovc\\_smpsoc\\_t](#) { [SMPSOC\\_50V](#) = 0x00U, [SMPSOC\\_52V](#) = 0x01U, [SMPSOC\\_54V](#) = 0x02U, [SMPSOC\\_56V](#) = 0x03U, [SMPSOC\\_58V](#) = 0x04U, [SMPSOC\\_60V](#) = 0x05U, [SMPSOC\\_62V](#) = 0x06U, [SMPSOC\\_64V](#) = 0x07U, [SMPSOC\\_66V](#) = 0x08U, [SMPSOC\\_68V](#) = 0x09U, [SMPSOC\\_70V](#) = 0x0AU, [SMPSOC\\_72V](#) = 0x0BU, [SMPSOC\\_74V](#) = 0x0CU, [SMPSOC\\_76V](#) = 0x0DU, [SMPSOC\\_78V](#) = 0x0EU, [SMPSOC\\_80V](#) = 0x0FU }  
*SMPS output voltage configuration options (0x1A:SMPSOC[3:0])*

#### Functions

- status\_t [UJA113X\\_SetSmpls](#) (uint32\_t instance, const [uja113x\\_smpsc\\_cfg\\_t](#) \*smpscConfig)  
*Sets the SMPS control registers (0x19 - 0x1A) based on input configuration object.*

### 16.11.3 Data Structure Documentation

#### 16.11.3.1 struct [uja113x\\_smpsc\\_cfg\\_t](#)

SMPS (buck-boost converter) registers configuration structure (0x19 - 0x1A)

Implements : [uja113x\\_smpsc\\_cfg\\_t\\_Class](#)

Definition at line 1515 of file [sbc\\_uja113x.h](#).

#### Data Fields

- [uja113x\\_smpsc\\_smpsotc\\_t](#) [smpsOvertempCtrl](#)  
*Configures the behavior of Vvsmps voltage output during overtemperature (0x19:SMPSOTC[3])*
- [uja113x\\_smpsc\\_smpsc\\_t](#) [smpsCtrl](#)  
*Configures the SMPS on/off state in different modes (0x19:SMPSOC[1:0])*
- [uja113x\\_smpsovc\\_smpsoc\\_t](#) [smpsOutVolt](#)  
*Configures the SMPS output voltage level (0x1A:SMPSOC[3:0])*

## Field Documentation

### 16.11.3.1.1 `uja113x_smpsc_smpsc_t smpsCtrl`

Configures the SMPS on/off state in different modes (0x19:SMPSOC[1:0])

Definition at line 1524 of file `sbc_uja113x.h`.

### 16.11.3.1.2 `uja113x_smpsovc_smpsoc_t smpsOutVolt`

Configures the SMPS output voltage level (0x1A:SMPSOC[3:0])

Definition at line 1528 of file `sbc_uja113x.h`.

### 16.11.3.1.3 `uja113x_smpsc_smpsotc_t smpsOvertempCtrl`

Configures the behavior of Vvsmps voltage output during overtemperature (0x19:SMPSOTC[3])

Definition at line 1520 of file `sbc_uja113x.h`.

## 16.11.4 Typedef Documentation

### 16.11.4.1 `typedef bool uja113x_smpsc_smpsotc_t`

SMPS overtemperature configuration options (0x19:SMPSOTC[3])

false - Vvsmps output not modified during overtemperature warning (OTWI interrupt)

true - Vvsmps output reduced to 5V when chip temperature is above the overtemperature warning threshold, T<sub>th(warn)otp</sub>

Implements : `uja113x_smpsc_smpsotc_t_Class`

Definition at line 468 of file `sbc_uja113x.h`.

## 16.11.5 Enumeration Type Documentation

### 16.11.5.1 `enum uja113x_smpsc_smpsc_t`

SMPS on/off configuration options (0x19:SMPSOC[1:0])

Implements : `uja113x_smpsc_smpsc_t_Class`

## Enumerator

***SMPSOC\_ON\_NSR*** SMPS is on in Normal, Standby and Reset modes and shut-down in all other modes

***SMPSOC\_ON\_NSRS*** SMPS is on in Normal, Standby, Reset and Sleep modes nad shut-down in all other modes

***SMPSOC\_PASS\_THROUGH\_NSS*** Pass-through mode is requested in Normal, Standby and Sleep modes

Definition at line 477 of file `sbc_uja113x.h`.

### 16.11.5.2 `enum uja113x_smpsovc_smpsoc_t`

SMPS output voltage configuration options (0x1A:SMPSOC[3:0])

Implements : `uja113x_smpsovc_smpsoc_t_Class`

## Enumerator

***SMPSOC\_50V*** SMPS output voltage 5.0 V

***SMPSOC\_52V*** SMPS output voltage 5.2 V

***SMPSOC\_54V*** SMPS output voltage 5.4 V

**SMPSOC\_56V** SMPS output voltage 5.6 V  
**SMPSOC\_58V** SMPS output voltage 5.8 V  
**SMPSOC\_60V** SMPS output voltage 6.0 V  
**SMPSOC\_62V** SMPS output voltage 6.2 V  
**SMPSOC\_64V** SMPS output voltage 6.4 V  
**SMPSOC\_66V** SMPS output voltage 6.6 V  
**SMPSOC\_68V** SMPS output voltage 6.8 V  
**SMPSOC\_70V** SMPS output voltage 7.0 V  
**SMPSOC\_72V** SMPS output voltage 7.2 V  
**SMPSOC\_74V** SMPS output voltage 7.4 V  
**SMPSOC\_76V** SMPS output voltage 7.6 V  
**SMPSOC\_78V** SMPS output voltage 7.8 V  
**SMPSOC\_80V** SMPS output voltage 8.0 V

Definition at line 494 of file sbc\_uja113x.h.

#### 16.11.6 Function Documentation

16.11.6.1 `status_t UJA113X_SetSmps ( uint32_t instance, const uja113x_smpsc_cfg_t * smpscConfig )`

Sets the SMPS control registers (0x19 - 0x1A) based on input configuration object.

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the SMPS control registers were not written properly and the process has to be repeated.

##### Parameters

in	<i>instance</i>	Index of SBC to configure
in	<i>smpscConfig</i>	SMPS control configuration object

##### Returns

STATUS\_SUCCESS Configuration successful  
 STATUS\_BUSY SPI peripheral is busy  
 STATUS\_TIMEOUT SPI communication timed out, configuration aborted  
 SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

##### Note

The function configures all the SMPS features:

1. SMPS overtemperature control (0x19)
2. SMPS on/off control (0x19)
3. SMPS output voltage (0x1A)

Definition at line 589 of file sbc\_uja113x.c.

## 16.12 03. Transceiver control

### 16.12.1 Detailed Description

The transceiver control registers (0x20 - 0x2F) allow:

- Configuring CAN (0x20)
- Configuring LIN1/2 (0x21)
- Read transceiver status (0x22)
- Enable transceiver interrupts (0x23)
- Configuring CAN partial networking
  - Data rate (0x26)
  - ID (0x27 - 0x2A)
  - ID mask (0x2B - 0x2E)
  - Frame control (0x2F)
  - Data masks (0x68 - 0x6F) (part of interrupt and status registers 0x60 - 0x6F)

Interrupts and interrupt status is described in [06. Interrupts](#).

#### Modules

- [01. CAN & CAN partial networking configuration](#)
- [02. LIN configuration](#)



## 16.13 04. HVIO bank 0 & 1 control

### 16.13.1 Detailed Description

### 16.13.2 Introduction

The HVIO control registers (0x30 - 0x3A / 0x40 - 0x4A) allow:

- Configuring the HVIO pin behaviour (0x30 - 0x33 / 0x40 - 0x43)
- Configuring the Bank N wake-up threshold (0x34 - 0x44)
- Read Bank N wake-up status (0x35 / 0x45)
- Read Bank N driver status (0x36 / 0x46)
- Configuring Bank N short-circuit detection threshold (0x39 / 0x49)
- Configuring Bank N open-load detection threshold (0x3A / 0x4A)
- Enabling/disabling HVIO N high-/low-side driver in non-volatile memory (0x71 / 0x72) (see [HVIO high-side configuration options \(0x71\)](#) & [HVIO low-side configuration options \(0x72\)](#) for details)

#### Note

Please, consult the details of HVIO capabilities with the reference manual, as this is out of the scope of this documentation.

#### 16.13.2.1 Short-circuit handling

Every HVIO pin can be configured separately in it's dedicated HVIO control register (0x30 - 0x33 / 0x40 - 0x43) by setting the IOnCC (HVIO configuration control) bit and IOnAC (HVIO activation control). The HVIO can be disabled by either setting the IOnAC bit to deactivated (see [uja113x\\_hvion\\_ionac\\_t](#)) or by IOnCC to HVIO off state (see [uja113x\\_hvion\\_ioncc\\_t](#)).

However, during short circuit, to reset the function of the HVIO pin, the IOnAC bit has to be set to deactivated and afterwards a corresponding active mode set again. Resetting the HVIO pin by IOnCC will have no effect!

##### 16.13.2.1.1 Example

```
/* Configuring the HVIO pin */
uja113x_hvio_cfg_t hvio1Config;
hvio1Config.ionShutdownCtrl = IONSC_OVERVOLTAGE;
hvio1Config.ionActivateCtrl = IONAC_ENABLED;
hvio1Config.ionConfCtrl = IONCC_HS_SLOPE;
UJA113X_SetHvio(SBC_INSTANCE_0, &hvio1Config, HVIO1);

/* Short-circuit detected on HVIO 1 */
hvio1Config.ionActivateCtrl = IONAC_DEACTIVATED;
UJA113X_SetHvio(SBC_INSTANCE_0, &hvio1Config, HVIO1);

/* delay/manage short-circuit etc. if necessary */
...
/* Activate HVIO again */
hvio1Config.ionActivateCtrl = IONAC_ENABLED;
UJA113X_SetHvio(SBC_INSTANCE_0, &hvio1Config, HVIO1);
```

#### Data Structures

- struct [uja113x\\_hvio\\_cfg\\_t](#)  
HVIO pin control register configuration structure (0x3N; 0x4N; N = 0 - 3) [More...](#)
- struct [uja113x\\_bscdt\\_cfg\\_t](#)  
Bank 0/1 short-circuit detection threshold configuration structure (0x39/0x49) [More...](#)
- struct [uja113x\\_bolddt\\_cfg\\_t](#)  
Bank 0/1 open-load detection threshold configuration structure (0x3A/0x4A) [More...](#)

## Typedefs

- typedef bool [uja113x\\_bwtc\\_bnwtc\\_t](#)  
*Bank N wake-up threshold configuration options (0x34 / 0x44)*

## Enumerations

- enum [uja113x\\_hvion\\_ionsc\\_t](#) { [IONSC\\_NONE](#) = 0x00U, [IONSC\\_UNDRVOLTAGE](#) = 0x40U, [IONSC\\_OVRVOLTAGE](#) = 0x80U, [IONSC\\_FULL](#) = 0xC0U }  
*HVION shut-down configuration options (0x3N:IONSC[7:6], N = 0, 1, 2, 3)*
- enum [uja113x\\_hvion\\_ionac\\_t](#) {  
[IONAC\\_DEACTIVATED](#) = 0x00U, [IONAC\\_ENABLED](#) = 0x08U, [IONAC\\_TMR\\_1](#) = 0x10U, [IONAC\\_TMR\\_2](#) = 0x18U,  
[IONAC\\_TMR\\_3](#) = 0x20U, [IONAC\\_TMR\\_4](#) = 0x28U, [IONAC\\_HVIO\\_INV](#) = 0x30U, [IONAC\\_HVIO](#) = 0x38U }  
*HVION activation configuration options (0x3N:IONAC[5:3]; N = 0, 1, 2, 3)*
- enum [uja113x\\_hvion\\_ioncc\\_t](#) {  
[IONCC\\_OFF](#) = 0x00U, [IONCC\\_HS\\_SLOPE](#) = 0x01U, [IONCC\\_LS\\_SLOPE](#) = 0x02U, [IONCC\\_WUP](#) = 0x03U,  
[IONCC\\_HS\\_WUP\\_SLOPE](#) = 0x04U, [IONCC\\_HS](#) = 0x05U, [IONCC\\_LS](#) = 0x06U, [IONCC\\_HS\\_WUP](#) = 0x07U  
}  
*HVION output configuration options (0x3N:IONCC[2:0]; N = 0, 1, 2, 3)*

## Functions

- status\_t [UJA113X\\_SetHvio](#) (uint32\_t instance, const [uja113x\\_hvio\\_cfg\\_t](#) \*hvioConfig, uint8\_t hvioId)  
*Configures corresponding HVIO control register based on hvioId (0x3N/0x4N; N = 0 - 3)*
- status\_t [UJA113X\\_SetBankWupThr](#) (uint32\_t instance, [uja113x\\_bwtc\\_bnwtc\\_t](#) bankWupThrConfig, uint8\_t bankId)  
*Configures corresponding bank wake-up control register based on bankId (0x34/0x44)*
- status\_t [UJA113X\\_SetBankShortCircThr](#) (uint32\_t instance, const [uja113x\\_bscdt\\_cfg\\_t](#) \*bscdtConfig, uint8\_t bankId)  
*Configures all HVIO's short-circuit detection threshold level based on bankId (0x39/0x49)*
- status\_t [UJA113X\\_SetBankOpenLoadThr](#) (uint32\_t instance, const [uja113x\\_boldt\\_cfg\\_t](#) \*boldtConfig, uint8\_t bankId)  
*Configures all HVIO's open-load detection threshold level based on bankId (0x3A/0x4A)*

## 16.13.3 Data Structure Documentation

### 16.13.3.1 struct [uja113x\\_hvio\\_cfg\\_t](#)

HVION control register configuration structure (0x3N; 0x4N; N = 0 - 3)

Implements : [uja113x\\_hvio\\_cfg\\_t\\_Class](#)

Definition at line 1646 of file [sbc\\_uja113x.h](#).

## Data Fields

- [uja113x\\_hvion\\_ionsc\\_t](#) [ionShutdownCtrl](#)  
*Configures HVION shut-down behavior (0x3N/0x4N:IONSC[7:6])*
- [uja113x\\_hvion\\_ionac\\_t](#) [ionActivateCtrl](#)  
*Configures HVION activation behavior (0x3N/0x4N:IONAC[5:3])*
- [uja113x\\_hvion\\_ioncc\\_t](#) [ionConfCtrl](#)  
*Configures HVION output behavior (0x3N/0x4N:IONCC[2:0])*

## Field Documentation

16.13.3.1.1 `uja113x_hvion_ionac_t ionActivateCtrl`

Configures HVIO activation behavior (0x3N/0x4N:[IOAC[5:3]])

Definition at line 1655 of file `sbc_uja113x.h`.

16.13.3.1.2 `uja113x_hvion_ioncc_t ionConfCtrl`

Configures HVIO output behavior (0x3N/0x4N:[IONCC[2:0]])

Definition at line 1659 of file `sbc_uja113x.h`.

16.13.3.1.3 `uja113x_hvion_ionsc_t ionShutdownCtrl`

Configures HVIO shut-down behavior (0x3N/0x4N:[IONSC[7:6]])

Definition at line 1651 of file `sbc_uja113x.h`.

16.13.3.2 `struct uja113x_bscdt_cfg_t`

Bank 0/1 short-circuit detection threshold configuration structure (0x39/0x49)

Implements : `uja113x_bscdt_cfg_t_Class`

Definition at line 1669 of file `sbc_uja113x.h`.

## Data Fields

- [uja113x\\_bscdt\\_ionsctc\\_t io48ShortCircuitThrCtrl](#)  
*Configures HVIO4(8) short-circuit detection threshold (0x39/0x49:IO4(8)SCTC[7:6])*
- [uja113x\\_bscdt\\_ionsctc\\_t io37ShortCircuitThrCtrl](#)  
*Configures HVIO3(7) short-circuit detection threshold (0x39/0x49:IO3(7)SCTC[5:4])*
- [uja113x\\_bscdt\\_ionsctc\\_t io26ShortCircuitThrCtrl](#)  
*Configures HVIO2(6) short-circuit detection threshold (0x39/0x49:IO2(6)SCTC[3:2])*
- [uja113x\\_bscdt\\_ionsctc\\_t io15ShortCircuitThrCtrl](#)  
*Configures HVIO1(5) short-circuit detection threshold (0x39/0x49:IO1(5)SCTC[1:0])*

## Field Documentation

16.13.3.2.1 `uja113x_bscdt_ionsctc_t io15ShortCircuitThrCtrl`

Configures HVIO1(5) short-circuit detection threshold (0x39/0x49:IO1(5)SCTC[1:0])

Definition at line 1686 of file `sbc_uja113x.h`.

16.13.3.2.2 `uja113x_bscdt_ionsctc_t io26ShortCircuitThrCtrl`

Configures HVIO2(6) short-circuit detection threshold (0x39/0x49:IO2(6)SCTC[3:2])

Definition at line 1682 of file `sbc_uja113x.h`.

16.13.3.2.3 `uja113x_bscdt_ionsctc_t io37ShortCircuitThrCtrl`

Configures HVIO3(7) short-circuit detection threshold (0x39/0x49:IO3(7)SCTC[5:4])

Definition at line 1678 of file `sbc_uja113x.h`.

16.13.3.2.4 `uja113x_bscdt_ionsctc_t io48ShortCircuitThrCtrl`

Configures HVIO4(8) short-circuit detection threshold (0x39/0x49:IO4(8)SCTC[7:6])

Definition at line 1674 of file `sbc_uja113x.h`.

### 16.13.3.3 struct uja113x\_boldt\_cfg\_t

Bank 0/1 open-load detection threshold configuration structure (0x3A/0x4A)

Implements : uja113x\_boldt\_cfg\_t\_Class

Definition at line 1696 of file sbc\_uja113x.h.

#### Data Fields

- [uja113x\\_boldt\\_ionoltc\\_t io48OpenLoadThrCtrl](#)  
*Configures HVIO4(8) open-load detection threshold (0x3A/4A:IO4(8)OLTC[7:6])*
- [uja113x\\_boldt\\_ionoltc\\_t io37OpenLoadThrCtrl](#)  
*Configures HVIO4(8) open-load detection threshold (0x3A/4A:IO3(7)OLTC[5:4])*
- [uja113x\\_boldt\\_ionoltc\\_t io26OpenLoadThrCtrl](#)  
*Configures HVIO4(8) open-load detection threshold (0x3A/4A:IO2(6)OLTC[3:2])*
- [uja113x\\_boldt\\_ionoltc\\_t io15OpenLoadThrCtrl](#)  
*Configures HVIO4(8) open-load detection threshold (0x3A/4A:IO1(5)OLTC[1:0])*

#### Field Documentation

##### 16.13.3.3.1 uja113x\_boldt\_ionoltc\_t io15OpenLoadThrCtrl

Configures HVIO4(8) open-load detection threshold (0x3A/4A:IO1(5)OLTC[1:0])

Definition at line 1713 of file sbc\_uja113x.h.

##### 16.13.3.3.2 uja113x\_boldt\_ionoltc\_t io26OpenLoadThrCtrl

Configures HVIO4(8) open-load detection threshold (0x3A/4A:IO2(6)OLTC[3:2])

Definition at line 1709 of file sbc\_uja113x.h.

##### 16.13.3.3.3 uja113x\_boldt\_ionoltc\_t io37OpenLoadThrCtrl

Configures HVIO4(8) open-load detection threshold (0x3A/4A:IO3(7)OLTC[5:4])

Definition at line 1705 of file sbc\_uja113x.h.

##### 16.13.3.3.4 uja113x\_boldt\_ionoltc\_t io48OpenLoadThrCtrl

Configures HVIO4(8) open-load detection threshold (0x3A/4A:IO4(8)OLTC[7:6])

Definition at line 1701 of file sbc\_uja113x.h.

### 16.13.4 Typedef Documentation

#### 16.13.4.1 typedef bool uja113x\_bwtc\_bnwtc\_t

Bank N wake-up threshold configuration options (0x34 / 0x44)

false - threshold is ratiometric to Vbaths1

true - threshold is absolute

Implements : uja113x\_bwtc\_bnwtc\_t\_Class

Definition at line 894 of file sbc\_uja113x.h.

### 16.13.5 Enumeration Type Documentation

#### 16.13.5.1 enum uja113x\_hvion\_ionac\_t

HVION activation configuration options (0x3N:IONAC[5:3]; N = 0, 1, 2, 3)

Implements : `uja113x_hvion_ionac_t_Class`

#### Enumerator

***IONAC\_DEACTIVATED*** HVION is deactivated.  
***IONAC\_ENABLED*** HVION is enabled.  
***IONAC\_TMR\_1*** HVION is controlled by timer 1.  
***IONAC\_TMR\_2*** HVION is controlled by timer 2.  
***IONAC\_TMR\_3*** HVION is controlled by timer 3.  
***IONAC\_TMR\_4*** HVION is controlled by timer 4.  
***IONAC\_HVIO\_INV*** HVION is controlled by HVION + 4 (inverted control; only available for bank 0)  
***IONAC\_HVIO*** HVION is controlled by HVION + 4 (non-inverted control; only available for bank 0)

Definition at line 854 of file `sbc_uja113x.h`.

#### 16.13.5.2 enum `uja113x_hvion_ioncc_t`

HVION output configuration options (0x3N:IONCC[2:0]; N = 0, 1, 2, 3)

Implements : `uja113x_hvion_ioncc_t_Class`

#### Enumerator

***IONCC\_OFF*** HVION is off.  
***IONCC\_HS\_SLOPE*** Configure HVION as high-side driver with slope control.  
***IONCC\_LS\_SLOPE*** Configure HVION as low-side driver with slope control.  
***IONCC\_WUP*** Configure HVION as wake-up input.  
***IONCC\_HS\_WUP\_SLOPE*** Configure HVION as high-side driver and wake-up input with slope control.  
***IONCC\_HS*** Configure HVION as high-side driver without slope control.  
***IONCC\_LS*** Configure HVION as low-side driver without slope control.  
***IONCC\_HS\_WUP*** Configure HVION as high-side driver and wake-up input without slope control.

Definition at line 873 of file `sbc_uja113x.h`.

#### 16.13.5.3 enum `uja113x_hvion_ionsc_t`

HVION shut-down configuration options (0x3N:IONSC[7:6], N = 0, 1, 2, 3)

Implements : `uja113x_hvion_ionsc_t_Class`

#### Enumerator

***IONSC\_NONE*** HVION doesn't respond to over- or undervoltage.  
***IONSC\_UNDRVOLTAGE*** HVION shuts down when battery undervoltage is detected in normal mode.  
***IONSC\_OVERVOLTAGE*** HVION shuts down when battery overvoltage is detected in normal mode.  
***IONSC\_FULL*** HVION shuts down when battery over- or undervoltage is detected in normal mode.

Definition at line 839 of file `sbc_uja113x.h`.

### 16.13.6 Function Documentation

16.13.6.1 `status_t UJA113X_SetBankOpenLoadThr ( uint32_t instance, const uja113x_boldt_cfg_t * boldtConfig, uint8_t bankId )`

Configures all HVIO's open-load detection threshold level based on bankId (0x3A/0x4A)

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the open-load detection threshold control register identified by bankId has not been written properly and the process has to be repeated.

**Parameters**

in	<i>instance</i>	Index of SBC to configure
in	<i>bscdtConfig</i>	Open-load threshold level configuration object
in	<i>bankId</i>	Bank identifier (0 - 1)

**Returns**

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

**Note**

The hviold parameter determines, which HVIO shall be configured.

Definition at line 1028 of file sbc\_uja113x.c.

**16.13.6.2** `status_t UJA113X_SetBankShortCircThr ( uint32_t instance, const uja113x_bscdt_cfg_t * bscdtConfig, uint8_t bankId )`

Configures all HVIO's short-circuit detection threshold level based on bankId (0x39/0x49)

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the short-circuit detection threshold control register identified by bankId has not been written properly and the process has to be repeated.

**Parameters**

in	<i>instance</i>	Index of SBC to configure
in	<i>bscdtConfig</i>	Short-circuit threshold level configuration object
in	<i>bankId</i>	Bank identifier (0 - 1)

**Returns**

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

**Note**

The bankId parameter determines, which bank shall be configured.

Definition at line 996 of file sbc\_uja113x.c.

**16.13.6.3** `status_t UJA113X_SetBankWupThr ( uint32_t instance, uja113x_bwtc_bnwtc_t bankWupThrConfig, uint8_t bankId )`

Configures corresponding bank wake-up control register based on bankId (0x34/0x44)

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the bank N wake-up control register identified by bankId has not been written properly and the process has to be repeated.

**Parameters**

in	<i>instance</i>	Index of SBC to configure
in	<i>bankWupThr↔ Config</i>	Bank N wake-up threshold configuration parameter
in	<i>bankId</i>	Bank identifier (0 - 1)

**Returns**

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

**Warning**

This function only checks the maximum number of HVIO banks, as write to "non-existing" registers are allowed. Thus for devices without HVIO this function will perform an "empty" write.

Definition at line 914 of file sbc\_uja113x.c.

**16.13.6.4** `status_t UJA113X_SetHvio ( uint32_t instance, const uja113x_hvio_cfg_t * hvioConfig, uint8_t hviold )`

Configures corresponding HVIO control register based on hviold (0x3N/0x4N; N = 0 - 3)

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the HVIO control register identified by hviold has not been written properly and the process has to be repeated.

**Parameters**

in	<i>instance</i>	Index of SBC to configure
in	<i>hvioConfig</i>	HVIO configuration object
in	<i>hviold</i>	HVIO identifier (0 = HVIO1 - 7 = HVIO8)

**Returns**

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

**Note**

The hviold parameter determines, which HVIO shall be configured.

Definition at line 877 of file sbc\_uja113x.c.

## 16.14 04. Lock control

### 16.14.1 Detailed Description

#### 16.14.2 Introduction

The Lock control allows the user to write protect specific memory areas. This way the SBC is protected against unintentional writes into the SBC registers providing additional level of safety/security.

An SPI failure interrupt is generated, if enabled (see [06. Interrupts](#)). Otherwise writes into locked areas are ignored.

- Lock 0: address area 0x06 - 0x09; general purpose memory
- Lock 1: address area 0x10 - 0x1F; supply control
- Lock 2: address area 0x20 - 0x2F; transceiver control
- Lock 3: address area 0x30 - 0x3F; HVIO1 - HVIO4 control
- Lock 4: address area 0x40 - 0x4F; HVIO5 - HVIO8 control
- Lock 5: address area 0x50 - 0x5F; timer control
- Lock 6: address area 0x68 - 0x6F; data mask (FD versions only)

#### Data Structures

- struct [uja113x\\_lckc\\_cfg\\_t](#)  
*Lock Control register configuration structure (reg 0x0A) [More...](#)*

#### Typedefs

- typedef bool [uja113x\\_lckc\\_lkxc\\_t](#)  
*Lock Control configuration options (0x0A)*

#### Functions

- status\_t [UJA113X\\_SetLock](#) (uint32\_t instance, const [uja113x\\_lckc\\_cfg\\_t](#) \*lckcConfig)  
*Sets the lock control register (0x0A) based on input configuration object.*

### 16.14.3 Data Structure Documentation

#### 16.14.3.1 struct uja113x\_lckc\_cfg\_t

Lock Control register configuration structure (reg 0x0A)

The structure defines, which address areas should be SPI write (un)protected.

#### See also

[UJA113X\\_SetLock\(\)](#)

Implements : [uja113x\\_lckc\\_cfg\\_t\\_Class](#)

Definition at line 1426 of file [sbc\\_uja113x.h](#).



## Data Fields

- [uja113x\\_lckc\\_lkxc\\_t lock6](#)  
*Disables/enables SPI write access to address area 0x68 - 0x6F (FD version only) (0x0A:LK6C[6]).*
- [uja113x\\_lckc\\_lkxc\\_t lock5](#)  
*Disables/enables SPI write access to address area 0x50 - 0x55 (0x0A:LK5C[5]).*
- [uja113x\\_lckc\\_lkxc\\_t lock4](#)  
*Disables/enables SPI write access to address area 0x40 - 0x4F (0x0A:LK4C[4]).*
- [uja113x\\_lckc\\_lkxc\\_t lock3](#)  
*Disables/enables SPI write access to address area 0x30 - 0x3F (0x0A:LK3C[3]).*
- [uja113x\\_lckc\\_lkxc\\_t lock2](#)  
*Disables/enables SPI write access to address area 0x20 - 0x2F (0x0A:LK2C[2]).*
- [uja113x\\_lckc\\_lkxc\\_t lock1](#)  
*Disables/enables SPI write access to address area 0x10 - 0x1F (0x0A:LK1C[1]).*
- [uja113x\\_lckc\\_lkxc\\_t lock0](#)  
*Disables/enables SPI write access to address area 0x06 - 0x09 (0x0A:LK0C[0]).*

## Field Documentation

16.14.3.1.1 [uja113x\\_lckc\\_lkxc\\_t lock0](#)

Disables/enables SPI write access to address area 0x06 - 0x09 (0x0A:LK0C[0]).

Definition at line 1440 of file sbc\_uja113x.h.

16.14.3.1.2 [uja113x\\_lckc\\_lkxc\\_t lock1](#)

Disables/enables SPI write access to address area 0x10 - 0x1F (0x0A:LK1C[1]).

Definition at line 1438 of file sbc\_uja113x.h.

16.14.3.1.3 [uja113x\\_lckc\\_lkxc\\_t lock2](#)

Disables/enables SPI write access to address area 0x20 - 0x2F (0x0A:LK2C[2]).

Definition at line 1436 of file sbc\_uja113x.h.

16.14.3.1.4 [uja113x\\_lckc\\_lkxc\\_t lock3](#)

Disables/enables SPI write access to address area 0x30 - 0x3F (0x0A:LK3C[3]).

Definition at line 1434 of file sbc\_uja113x.h.

16.14.3.1.5 [uja113x\\_lckc\\_lkxc\\_t lock4](#)

Disables/enables SPI write access to address area 0x40 - 0x4F (0x0A:LK4C[4]).

Definition at line 1432 of file sbc\_uja113x.h.

16.14.3.1.6 [uja113x\\_lckc\\_lkxc\\_t lock5](#)

Disables/enables SPI write access to address area 0x50 - 0x55 (0x0A:LK5C[5]).

Definition at line 1430 of file sbc\_uja113x.h.

16.14.3.1.7 [uja113x\\_lckc\\_lkxc\\_t lock6](#)

Disables/enables SPI write access to address area 0x68 - 0x6F (FD version only) (0x0A:LK6C[6]).

Definition at line 1428 of file sbc\_uja113x.h.

#### 16.14.4 Typedef Documentation

##### 16.14.4.1 typedef bool `uja113x_lckc_lkxc_t`

Lock Control configuration options (0x0A)

false - do not lock memory; SPI write access enabled

true - lock memory; SPI write access disabled

Implements : `uja113x_lckc_lkxc_t_Class`

Definition at line 364 of file `sbc_uja113x.h`.

#### 16.14.5 Function Documentation

##### 16.14.5.1 `status_t UJA113X_SetLock ( uint32_t instance, const uja113x_lckc_cfg_t * lckcConfig )`

Sets the lock control register (0x0A) based on input configuration object.

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the lock control register has not been written and the process has to be repeated.

##### Parameters

<i>in</i>	<i>instance</i>	Index of SBC to configure
<i>in</i>	<i>lckcConfig</i>	Lock control configuration object

##### Returns

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

##### Note

This function sets a lock on certain memory areas to prevent unintended register modifications over SPI, thus the function should be called after the whole SBC has been initialized!

Write accesses to locked memory areas can lead to SPI failure interrupt, if enabled.

The lock register is never locked, thus future (un)locking is possible.

Definition at line 488 of file `sbc_uja113x.c`.

## 16.15 05. Timer control

### 16.15.1 Detailed Description

#### 16.15.2 Introduction

Devices incorporating HVIOs functionality contain 4 timers. Up to four configuration options are available (see [uja113x\\_timxc\\_txmc\\_t](#)). Configuring a single timer can be achieved by invoking [UJA113X\\_SetTimer\(\)](#). To change just the duty cycle, the [UJA113X\\_SbcTransfer\(\)](#) can be used. For more details, consult the reference manual.

#### Data Structures

- struct [uja113x\\_timsc\\_cfg\\_t](#)

*Single timer control registers configuration structure (0x5x - 0x5x + 1) [More...](#)*

#### Typedefs

- typedef uint8\_t [uja113x\\_timxdcc\\_txdcc\\_t](#)

*Timer X duty cycle configuration options (0x5N:TDCC[7:0]); (N = 1, 3, 5, 7)*

#### Enumerations

- enum [uja113x\\_timxc\\_txpc\\_t](#) {  
[TXPC\\_4MS\\_PERIOD](#) = 0x00U, [TXPC\\_8MS\\_PERIOD](#) = 0x04U, [TXPC\\_20MS\\_PERIOD](#) = 0x08U, [TXPC\\_30MS\\_PERIOD](#) = 0x0CU,  
[TXPC\\_50MS\\_PERIOD](#) = 0x10U, [TXPC\\_100MS\\_PERIOD](#) = 0x14U, [TXPC\\_200MS\\_PERIOD](#) = 0x18U, [TXPC\\_400MS\\_PERIOD](#) = 0x1CU,  
[TXPC\\_800MS\\_PERIOD](#) = 0x20U, [TXPC\\_1000MS\\_PERIOD](#) = 0x24U, [TXPC\\_2000MS\\_PERIOD](#) = 0x28U,  
[TXPC\\_4000MS\\_PERIOD](#) = 0x2CU }

*Timer X period configuration options (0x5N:TXPC[5:2]); (N = 0, 2, 4, 6)*

- enum [uja113x\\_timxc\\_txmc\\_t](#) { [TXMC\\_PWM](#) = 0x00U, [TXMC\\_TIM](#) = 0x01U, [TXMC\\_TRIG\\_START](#) = 0x02U, [TXMC\\_TRIG\\_END](#) = 0x03U }

*Timer X mode control configuration options (0x5N:TXMC[0]); (N = 0, 2, 4, 6)*

#### Functions

- status\_t [UJA113X\\_SetTimer](#) (uint32\_t instance, const [uja113x\\_timsc\\_cfg\\_t](#) \*timscConfig)

*Sets a single timer configuration registers (0x5N - 0x5M) based on input configuration object (N = 0, 2, 4, 6; M = 1, 3, 5, 7)*

### 16.15.3 Data Structure Documentation

#### 16.15.3.1 struct uja113x\_timsc\_cfg\_t

Single timer control registers configuration structure (0x5x - 0x5x + 1)

Implements : [uja113x\\_timsc\\_cfg\\_t\\_Class](#)

Definition at line 1724 of file [sbc\\_uja113x.h](#).

## Data Fields

- [uint8\\_t nTimer](#)  
*Timer to configure (0 - 3)*
- [uja113x\\_timxc\\_tnpc\\_t timXPeriodCtrl](#)  
*Configures Timer X period (reg 0x5N:TXPC[5:2]; N = 0, 2, 4, 6)*
- [uja113x\\_timxc\\_txmc\\_t timXModeCtrl](#)  
*Configures Timer X mode (reg 0x5N:TXMC[1:0]); N = 0, 2, 4, 6)*
- [uja113x\\_timxdcc\\_txdcc\\_t timXDutyCycleCtrl](#)  
*Configures Timer X duty cycle (reg 0x5N:TXDCC[7:0]; N = 1, 3, 5, 7)*

## Field Documentation

### 16.15.3.1.1 uint8\_t nTimer

Timer to configure (0 - 3)

Definition at line 1729 of file sbc\_uja113x.h.

### 16.15.3.1.2 uja113x\_timxdcc\_txdcc\_t timXDutyCycleCtrl

Configures Timer X duty cycle (reg 0x5N:TXDCC[7:0]; N = 1, 3, 5, 7)

**duty cycle** = TXDCC / 255 x 100 [%]

Definition at line 1742 of file sbc\_uja113x.h.

### 16.15.3.1.3 uja113x\_timxc\_txmc\_t timXModeCtrl

Configures Timer X mode (reg 0x5N:TXMC[1:0]); N = 0, 2, 4, 6)

Definition at line 1737 of file sbc\_uja113x.h.

### 16.15.3.1.4 uja113x\_timxc\_tnpc\_t timXPeriodCtrl

Configures Timer X period (reg 0x5N:TXPC[5:2]; N = 0, 2, 4, 6)

Definition at line 1733 of file sbc\_uja113x.h.

## 16.15.4 Typedef Documentation

### 16.15.4.1 typedef uint8\_t uja113x\_timxdcc\_txdcc\_t

Timer X duty cycle configuration options (0x5N:TXDCC[7:0]); (N = 1, 3, 5, 7)

Implements : uja113x\_timxdcc\_txdcc\_t\_Class0 - 255;

**duty cycle** = TXDCC \ 255

Definition at line 1120 of file sbc\_uja113x.h.

## 16.15.5 Enumeration Type Documentation

### 16.15.5.1 enum uja113x\_timxc\_txmc\_t

Timer X mode control configuration options (0x5N:TXMC[0]); (N = 0, 2, 4, 6)

## Warning

Last two options (TXMC\_TRIG\_START and TXMC\_TRIG\_END) are not valid for Timer 1!

Implements : uja113x\_timxc\_txmc\_t\_Class

## Enumerator

**TXMC\_PWM** Timer N is in PWM mode; **on-time** =  $\text{TXDCC} \times \text{TXPC} / 255$ .

**TXMC\_TIM** Timer N is in Timer mode; **on-time** =  $\text{TXDCC} \times \text{tw}(\text{base})\text{tmr us}$ .

**TXMC\_TRIG\_START** Timer N pulse is triggered at the start of Timer 1 pulse (master-slave mode); **on-time** =  $\text{TXDCC} \times \text{tw}(\text{base})\text{tmr us}$ .

**TXMC\_TRIG\_END** Timer N pulse is triggered at the end of Timer 1 pulse (master-slave mode); **on-time** =  $\text{TXDCC} \times \text{tw}(\text{base})\text{tmr us}$ .

Definition at line 1105 of file sbc\_uja113x.h.

## 16.15.5.2 enum uja113x\_timxc\_txpc\_t

Timer X period configuration options (0x5N:TXPC[5:2]); (N = 0, 2, 4, 6)

Implements : uja113x\_timxc\_txpc\_t\_Class

## Enumerator

**TXPC\_4MS\_PERIOD** Timer period 4 ms

**TXPC\_8MS\_PERIOD** Timer period 8 ms

**TXPC\_20MS\_PERIOD** Timer period 20 ms

**TXPC\_30MS\_PERIOD** Timer period 30 ms

**TXPC\_50MS\_PERIOD** Timer period 50 ms

**TXPC\_100MS\_PERIOD** Timer period 100 ms

**TXPC\_200MS\_PERIOD** Timer period 200 ms

**TXPC\_400MS\_PERIOD** Timer period 400 ms

**TXPC\_800MS\_PERIOD** Timer period 800 ms

**TXPC\_1000MS\_PERIOD** Timer period 1 s

**TXPC\_2000MS\_PERIOD** Timer period 2 s

**TXPC\_4000MS\_PERIOD** Timer period 4 s

Definition at line 1080 of file sbc\_uja113x.h.

## 16.15.6 Function Documentation

## 16.15.6.1 status\_t UJA113X\_SetTimer ( uint32\_t instance, const uja113x\_timsc\_cfg\_t \* timscConfig )

Sets a single timer configuration registers (0x5N - 0x5M) based on input configuration object (N = 0, 2, 4, 6; M = 1, 3, 5, 7)

The function prepares a frame based on input configuration object and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the timer configuration registers have not been written properly and the process has to be repeated.

## Parameters

in	<i>instance</i>	Index of SBC to configure
in	<i>timscConfig</i>	Single timer configuration object

## Returns

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

**Return values**

<i>SBC_ERR_NA</i>	Timer with provided index is not present on this device.
-------------------	--

**Note**

This function configures a single timer. In case the user is interested in configuring the period or duty cycle only, [UJA113X\\_SbcTransfer\(\)](#) be used instead!

**Warning**

For Timer 1 the `timXPeriodCtrl` parameter is masked by 0x01! See [uja113x\\_timxc\\_txmc\\_t](#) for details!  
This function only checks the maximum number of timer channels, as write to "non-existing" registers are allowed. Thus for devices without timers this function will perform an "empty" write.

**See also**

`UJA113X_WriteRegister()`

Definition at line 1059 of file `sbc_uja113x.c`.

## 16.16 06. Interrupts

### 16.16.1 Detailed Description

#### 16.16.2 Introduction

The library provides one function for each interrupt group ([UJA113X\\_SetSystemInterrupts\(\)](#), [UJA113X\\_SetSupplyInterrupts\(\)](#) etc.) to enable/disable required interrupts. On the other hand, handling the interrupt's is not supported and it's the users responsibility to provide his own logic.

##### 16.16.2.1 Handling pending interrupts

After an event related to enabled interrupt occurs, the corresponding pin INTN1/INTN2 is set low. This can be detected by the MCU and a pending interrupt can be processed.

The SBC provides a global interrupt status register (see [Register address map](#)). This should be read first, to identify the interrupt group source. Afterwards a dedicated interrupt status register should be read to identify the interrupt source. The [UJA113X\\_SbcTransfer\(\)](#) function can be used.

#### Modules

- [Bank N fail interrupt enable configuration options \(0x38/0x48\)](#)
- [Bank N wake-up interrupt enable configuration options \(0x37/0x47\)](#)
- [Supply interrupt enable configuration options \(0x1C\)](#)
- [System interrupt enable configuration options \(0x04\)](#)
- [Transceiver interrupt enable configuration options \(0x23\)](#)

#### Functions

- `status_t UJA113X_SetSystemInterrupts (uint32_t instance, uint8_t sysieMask)`  
*Sets the system interrupt enable register (0x04) based on input configuration parameter.*
- `status_t UJA113X_SetSupplyInterrupts (uint32_t instance, uint8_t supieMask)`  
*Sets the supply interrupt enable register (0x1C) based on input parameter.*
- `status_t UJA113X_SetTransceiverInterrupts (uint32_t instance, uint8_t tieMask)`  
*Sets the transceiver interrupt enable register (0x23) based on input configuration parameter.*
- `status_t UJA113X_SetBankWupInterrupts (uint32_t instance, uint8_t bankWupIntMask, uint8_t bankId)`  
*Configures corresponding bank wake-up interrupt enable register based on bankId (0x37 / 0x47)*
- `status_t UJA113X_SetBankFailInterrupts (uint32_t instance, uint8_t bankFailIntMask, uint8_t bankId)`  
*Configures corresponding bank fail interrupt enable register based on bankId (0x38 / 0x48)*

### 16.16.3 Function Documentation

#### 16.16.3.1 `status_t UJA113X_SetBankFailInterrupts ( uint32_t instance, uint8_t bankFailIntMask, uint8_t bankId )`

Configures corresponding bank fail interrupt enable register based on bankId (0x38 / 0x48)

The function prepares a frame based on input configuration parameter and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the bank N fail interrupt enable register identified by bankId has not been written properly and the process has to be repeated.

#### Parameters

in	<i>instance</i>	Index of SBC to configure
in	<i>bankFailIntMask</i>	Mask defining interrupts to be enabled
in	<i>bankId</i>	Bank identifier (0 - 1)

#### Returns

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

#### See also

[Bank N fail interrupt enable configuration options \(0x38/0x48\)](#)

Definition at line 969 of file sbc\_uja113x.c.

**16.16.3.2** `status_t UJA113X_SetBankWupInterrupts ( uint32_t instance, uint8_t bankWupIntMask, uint8_t bankId )`

Configures corresponding bank wake-up interrupt enable register based on bankId (0x37 / 0x47)

The function prepares a frame based on input configuration parameter and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the bank N wake-up interrupt enable register identified by bankId has not been written properly and the process has to be repeated.

#### Parameters

in	<i>instance</i>	Index of SBC to configure
in	<i>bankWupIntMask</i>	Mask defining interrupts to be enabled
in	<i>bankId</i>	Bank identifier (0 - 1)

#### Returns

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

#### Warning

This function only checks the maximum number of HVIO banks, as write to "non-existing" registers are allowed. Thus for devices without HVIO this function will perform an "empty" write.

#### See also

[Bank N wake-up interrupt enable configuration options \(0x37/0x47\)](#)

Definition at line 942 of file sbc\_uja113x.c.

**16.16.3.3** `status_t UJA113X_SetSupplyInterrupts ( uint32_t instance, uint8_t supieMask )`

Sets the supply interrupt enable register (0x1C) based on input parameter.

The function prepares a frame based on input configuration parameter and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the supply interrupt register has not been written properly and the process has to be repeated.



**Parameters**

in	<i>instance</i>	Index of SBC to configure
in	<i>supieMask</i>	Defines which supply interrupts to enable

**Returns**

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

**See also**

[Supply interrupt enable configuration options \(0x1C\)](#)

Definition at line 617 of file sbc\_uja113x.c.

#### 16.16.3.4 status\_t UJA113X\_SetSystemInterrupts ( uint32\_t *instance*, uint8\_t *sysieMask* )

Sets the system interrupt enable register (0x04) based on input configuration paramater.

The function prepares a frame based on input configuration parameter and initiates SPI communcation in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the system interrupt enable register has not been written and the process has to be repeated.

Bits 7 - 3 & 0 are cleared (reserved bits).

**Parameters**

in	<i>instance</i>	Index of SBC to configure
in	<i>sysieMask</i>	Defines which system interrupts to enable

**Returns**

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

**See also**

[System interrupt enable configuration options \(0x04\)](#)

Definition at line 464 of file sbc\_uja113x.c.

#### 16.16.3.5 status\_t UJA113X\_SetTransceiverInterrupts ( uint32\_t *instance*, uint8\_t *tieMask* )

Sets the transceiver interrupt enable register (0x23) based on input configuration parameter.

The function prepares a frame based on input configuration parameter and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the transceiver interrupt enable register has not been written properly and the process has to be repeated.

Bits 7 - 5 are cleared (reserved bits).

**Parameters**

in	<i>instance</i>	Index of SBC to configure
in	<i>tieMask</i>	Defines which transceiver interrupts to enable

**Returns**

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI peripheral is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address, configuration failed.

See also

[Transceiver interrupt enable configuration options \(0x23\)](#)

Definition at line 722 of file sbc\_uja113x.c.

## 16.17 07. MTPNV and ID

### 16.17.1 Detailed Description

#### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\middleware\sbc\sbc_uja113x\source\sbc_uja113x.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\middleware\sbc\sbc_uja113x\include
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

[Low Power Serial Peripheral Interface \(LPSPi\) OS Interface \(OSiF\)](#)

### 16.17.2 Introduction

The UJA113x devices incorporate non-volatile memory for SBC configuration. This memory is not reset during off state and influences the chip behavior:

- Register 0x71
  - HVIOs high-side driver control
- Register 0x72
  - HVIOs low-side driver control
- Register 0x73
  - Reset pulse width
  - V2 regulator start-up
  - HVIO 4 - 2 mode
- Register 0x74
  - V1 reset threshold at start-up
  - FNMC mode/debug feature control
  - SDMC mode/debug feature control
  - VEXTAC pin
  - Sleep mode activation

During SW development, to simplify the application debugging, the FNMC and SDMC modes can be used to (partially) disable the Watchdog. In production, the FNMC and SDMC should be disabled to have always full Watchdog functionality present! However, it could be demanding to restore the MTPNV back to factory settings (FNMC enabled) and configure the MTPNV parameters again, but this time with SDMC mode enabled for diagnostics purposes. Or simply enable debug feature by configuring the watchdog into autonomous mode.

Once a UJA113X device has been registered, the [UJA113X\\_SetMtpnv\(\)](#) can be invoked. This function is called with the parameter set of parameters to be programmed into the non-volatile memory.

This allows the user to define production parameters with FNMC & SDMC disabled and in case of need to enable it again without the need of reflashing the host MCU with some debug firmware.

### 16.17.2.1 Restoring MTPNV factory preset values

The values are restored when following conditions apply for at least `td(MTPNV)` during power-up:

- pin RSTN is held LOW
- CANH is pulled up to Vbat
- CANL is pulled down to GND

#### Warning

Restoring the devices MTPNV back to factory preset values is not supported by the library and it's the users responsibility to implement this feature either in SW or HW.

### 16.17.2.2 MTPNV debug flow control

To have the possibility to enable debugging features (through SDMC mode and watchdog autonomous mode) but keep the MCUs firmware untouched, an MTPNV control sequence depicted on the following figure is recommended: This mechanism allows to partially or completely disable the watchdog if required without the need of flashing a custom firmware into the host MCU. In case the MTPNV has been already programmed or it's not allowed by the application to clear the MTPNV, it's still possible to switch the watchdog mode into autonomous, which keeps the watchdog enabled only in normal mode.

#### Note

If it's requested to set the SDMC mode, the MTPNV has to be restored prior to the above procedure (not depicted in the diagram).

The idea is:

- Reprogramming the MTPNV to enable SDMC
  - 1. Restore factory settings of MTPNV (as described earlier)
  - 2. Program the MTPNV with SDMC enabled
  - 3. Depending on situation switch the watchdog mode to autonomous
  - 4. The watchdog is disabled all the time
- Keep the MTPNV untouched (SDMC disabled) and partially disable watchdog
  - 1. Switch the Watchdog mode to autonomous
  - 2. Watchdog disabled in all other modes then normal (in case there are no pending interrupts)

#### Note

See [02. Watchdog](#) section for more details on watchdog behaviour under different circumstances

### 16.17.2.3 Example

This example demonstrates one of many possible implementations.

```
.
.
/* Define the MTPNV parameters */
ujall3x_mtpnv_cfg_t mtpnvConfig =
{
    /* highSideCtrl */
    HHSDC_IO8HOC_DISABLED | HHSDC_IO7HOC_DISABLED |
    HHSDC_IO6HOC_DISABLED | HHSDC_IO5HOC_DISABLED |
```

```

HNSDC_IO4HOC_DISABLED | HNSDC_IO3HOC_DISABLED |
HNSDC_IO2HOC_DISABLED | HNSDC_IO1HOC_DISABLED,

/* lowSideCtrl */
HNSDC_IO8LOC_DISABLED | HNSDC_IO7LOC_DISABLED |
HNSDC_IO6LOC_DISABLED | HNSDC_IO5LOC_DISABLED |
HNSDC_IO4LOC_DISABLED | HNSDC_IO3LOC_DISABLED |
HNSDC_IO2LOC_DISABLED | HNSDC_IO1LOC_DISABLED,

/* startupCtrl */
RLC_PULSE_WIDTH_10_12_MS | V2SUC_ALWAYS_ON |
IO4SFC_STD_IO | IO3SFC_STD_IO | IO2SFC_LIMP_HOME,

/* sbcConfigCtrl */
V1RTSUC_90_NOM | FNMC_DISABLE | SDMC_ENABLE |
VEXTAC_SENSOR_SUPPLY | SLPC_SLEEP_SUPPORTED
}
.
.

#define SBC0 0U

void main (void)
{
    /* Initialize the MCU - clock, peripherals etc. */
    ...
    /* Initialize the SPI periheral for SBC */
    ...
    /* Initialize the SBC */
    uja113x_drv_cfg_t sbcConfig =
    {
        UJA1132HW5V0, /* chipType */
        0U,           /* SPI instance */
        100U,         /* SPI timeout */
    };

    UJA113X_RegisterDevice(SBC0, &sbcConfig);

    bool enable_debug_mode = false;
    bool set_debug         = false;

    while (1U)
    {
        /*
         * Set enable_debug_mode and set_debug using some event over
         * CAN/LIN/USART etc. or GPIO inputs - application specific
         */
        ...

        if (true == enable_debug_mode)
        {
            /* Read the NVMPs status bit - UJA113X_SbcTransfer() can be used */
            ...

            if (1U == NVMPs)
            {
                /* Programm the MTPNV */
                UJA113X_SetMtpnv(SBC0, mtpnvConfig);
            }
            else
            {
                if (true == set_debug)
                {
                    /* Set Watchdog into autonomous mode - UJA113X_SetWatchdog() can be used */
                    ...
                }
                else
                {
                    /* Set Watchdog into application defined mode - UJA113X_SetWatchdog() can be used*/
                    ...
                }
            }
        }
    }
}

```

#### 16.17.2.4 CRC calculation on MTPNV data

To confirm MTPNV write, last byte written into the SBC (register 0x75) has to be a CRC-8 computed over the written bytes (4 bytes!). This is handled automatically by the [UJA113X\\_SetMtpnv\(\)](#). However if the user wants to implemet his own programming procedure, the CRC function is available, see [UJA113X\\_ComputeCrc8\(\)](#) for details.

**Note**

During CRC computation for devices with partial/none HVIO's the respective bits in registers 0x71 & 0x72 should be considered as 0 even when set by user to 1!

**16.17.2.5 Device ID**

Although, there is no dedicated function for reading the devices ID, the [UJA113X\\_SbcTransfer\(\)](#) can be used to read the ID registers (0x7E & 0x7F).

**Modules**

- [HVIO high-side configuration options \(0x71\)](#)
- [HVIO low-side configuration options \(0x72\)](#)
- [SBC configuration control options \(0x74\)](#)
- [Start-up control configuration options \(0x73\)](#)

**Data Structures**

- struct [uja113x\\_mtpnv\\_cfg\\_t](#)  
*MTPNV registers configuration structure (0x71 - 0x74) [More...](#)*
- struct [crc\\_spec\\_t](#)  
*Defines the CRC8 algorithm to be used. [More...](#)*

**Functions**

- status\_t [UJA113X\\_SetMtpnv](#) (uint32\_t instance, const [uja113x\\_mtpnv\\_cfg\\_t](#) \*mtpnvConfig)  
*Writes the non-volatile memory parameters (0x71 - 0x75) of SBC.*
- uint8\_t [UJA113X\\_ComputeCrc8](#) (const [crc\\_spec\\_t](#) \*crcDesc, const uint8\_t \*data, uint8\_t size)  
*Generic function for CRC8 computation.*

**16.17.3 Data Structure Documentation****16.17.3.1 struct uja113x\_mtpnv\_cfg\_t**

MTPNV registers configuration structure (0x71 - 0x74)

Implements : [uja113x\\_mtpnv\\_cfg\\_t\\_Class](#)

Definition at line 1752 of file [sbc\\_uja113x.h](#).

**Data Fields**

- uint8\_t [highSideCtrl](#)  
*Configures the HVIO high-side driver behavior (0x71); See [HVIO high-side configuration options \(0x71\)](#).*
- uint8\_t [lowSideCtrl](#)  
*Configures the HVIO low-side driver behavior (0x72); See [HVIO low-side configuration options \(0x72\)](#).*
- uint8\_t [startupCtrl](#)  
*Configures the start-up control register (0x73); See [Start-up control configuration options \(0x73\)](#).*
- uint8\_t [sbcConfigCtrl](#)  
*Configures the SBC configuration control register (0x74); See [SBC configuration control options \(0x74\)](#).*

## Field Documentation

## 16.17.3.1.1 uint8\_t highSideCtrl

Configures the HVIO high-side driver behavior (0x71); See [HVIO high-side configuration options \(0x71\)](#).

Definition at line 1757 of file sbc\_uja113x.h.

## 16.17.3.1.2 uint8\_t lowSideCtrl

Configures the HVIO low-side driver behavior (0x72); See [HVIO low-side configuration options \(0x72\)](#).

Definition at line 1761 of file sbc\_uja113x.h.

## 16.17.3.1.3 uint8\_t sbcConfigCtrl

Configures the SBC configuration control register (0x74); See [SBC configuration control options \(0x74\)](#).

Definition at line 1769 of file sbc\_uja113x.h.

## 16.17.3.1.4 uint8\_t startupCtrl

Configures the start-up control register (0x73); See [Start-up control configuration options \(0x73\)](#).

Definition at line 1765 of file sbc\_uja113x.h.

## 16.17.3.2 struct crc\_spec\_t

Defines the CRC8 algorithm to be used.

This structure describes the CRC algorithm. Based on this structure the [UJA113X\\_ComputeCrc8\(\)](#), knows, how to compute the CRC value.

Implements : `crc_spec_t_Class`

Definition at line 2638 of file sbc\_uja113x.h.

## Data Fields

- uint8\_t [poly](#)  
*Generating polynomial.*
- uint8\_t [init](#)  
*Initial value.*
- bool [refIn](#)  
*Reflect input data before computation.*
- bool [refOut](#)  
*Reflect output data before computation.*
- uint8\_t [xorOut](#)  
*Output xored by this value.*

## Field Documentation

## 16.17.3.2.1 uint8\_t init

Initial value.

Definition at line 2641 of file sbc\_uja113x.h.

## 16.17.3.2.2 uint8\_t poly

Generating polynomial.

Definition at line 2640 of file sbc\_uja113x.h.

### 16.17.3.2.3 bool refln

Reflect input data before computation.

Definition at line 2642 of file sbc\_uja113x.h.

### 16.17.3.2.4 bool refOut

Reflect output data before computation.

Definition at line 2643 of file sbc\_uja113x.h.

### 16.17.3.2.5 uint8\_t xorOut

Output xored by this value.

Definition at line 2644 of file sbc\_uja113x.h.

## 16.17.4 Function Documentation

### 16.17.4.1 uint8\_t UJA113X\_ComputeCrc8 ( const crc\_spec\_t \* crcDesc, const uint8\_t \* data, uint8\_t size )

Generic function for CRC8 computation.

The CRC algorithm is defined by `crc_spec_t` input parameter, which holds all the relevant information for CRC8 computation. A brute force mechanism is used instead of table driven.

#### Parameters

in	<i>crcDesc</i>	Pointer to CRC8 algorithm description
in	<i>data</i>	Array of data over which CRC should be calculated
in	<i>size</i>	Size of array in bytes

#### Returns

Computed CRC8 over *data* described by *crcDesc*

#### Note

For subsequent calls the CRC from previous computation should be used as initial seed.

#### Warning

The *refln* and *refOut* values have no influence on the computation! The functionality is not implemented yet!

### 16.17.4.2 status\_t UJA113X\_SetMtpnv ( uint32\_t instance, const uja113x\_mtpnv\_cfg\_t \* mtpnvConfig )

Writes the non-volatile memory parameters (0x71 - 0x75) of SBC.

The function prepares a frame based on input configuration parameters and initiates SPI communication in blocking mode. On successful transmission the first byte is verified. In case of a mismatch the MTPNV configuration registers have not been written properly and the process has to be repeated. In case a mismatch happens on CRC write, the outcome can be recognized by reading out the MTPNV registers, reading the MTPNV status bit and an SBC reset in case of success.



**Parameters**

in	<i>instance</i>	Index of SBC to configure
in	<i>mtpnvConfig</i>	Parameters to be loaded into SBC

**Returns**

STATUS\_SUCCESS Configuration successful STATUS\_BUSY SPI perihperal is busy STATUS\_TIMEOUT SPI communication timed out, configuration aborted SBC\_CMD\_ERROR First received byte doesn't match the repeated address,

**Note**

The CRC value is calculated by the function.

**Warning**

During CRC computation for devices with partial/none HVIO's the respective bits in registers 0x71 & 0x72 should be considered as 0 even when set by user to 1! The function checks the chipType registered with [UJA113X\\_RegisterDevice\(\)](#) and cleares these bits for the purpose of CRC computation.

Definition at line 1091 of file sbc\_uja113x.c.

## 16.18 ADC Driver

### 16.18.1 Detailed Description

Analog to Digital Converter Peripheral Driver.

The ADC is a configurable 12-bit (selectable to between 8-bit, 10-bit and 12-bit resolution) single-ended SAR converter.

Features of the ADC include:

- up to 32 control channels (depending on the device variant), with configurable triggers
- up to 32 selectable external input sources (depending on the device variant) and multiple internal input sources
- hardware compare and average functions
- auto-calibration feature

#### Hardware background

The ADC included in the S32K14x series is a selectable resolution (8, 10, 12-bit), single-ended, SAR converter. Depending on the device variant, each ADC instance has up to 40 selectable input channels (up to 32 external and up to 8 internal) and up to 32 control channels (each with a result register, an input channel selection register and interrupt enable).

**Sample time** is configurable through selection of A/D clock and a configurable sample time (in A/D clocks).

Also provided are the Hardware Average and Hardware Compare Features.

**Hardware Average** will sample a selectable number of measurements and average them before signaling a Conversion Complete.

**Hardware Compare** can be used to signal if an input channel goes outside (or inside) of a predefined range.

The **Calibration** features can be used to automatically calibrate or fine-tune the ADC before use.

#### Driver consideration

The ADC Driver provides access to all features, but not all need to be configured to use the ADC. The user application can use the default for most settings, changing only what is necessary. For example, if Compare or Average features are not used, the user does not need to configure them.

The Driver uses structures for configuration. Each structure contains members that are specific to its respective functionality. There is a **converter** structure, a hardware **compare** structure, a hardware **average** structure and a **calibration** structure. Each struct has a corresponding `InitStruct()` method that can be used to initialize the members to reset values, so the user can change only the values that are specific to the application.

The Driver also includes support for configuring the Trigger Latching and Arbitration Unit controlled from a separate hardware module - System Integration Module (SIM).

#### Interrupt handling

The ADC Driver in S32 SDK does not use interrupts internally. These can be defined by the user application. There are two ways to add an ADC interrupt:

1. Using the weak symbols defined by start-up code. If the methods `ADCx_Handler(void)` (x denotes instance number) are not defined, the linker uses a default ISR. An error will be generated if methods with the same name are defined multiple times. This method works regardless of the placement of the interrupt vector table (Flash or RAM).
2. Using the Interrupt Manager's `INT_SYS_InstallHandler()` method. This can be used to dynamically change the ISR at run-time. This method works only if the interrupt vector table is located in RAM (S32 SDK behavior). To get the ADC instance's interrupt number, use `ADC_DRV_GetInterruptNumber()`.

### Clocking and pin configuration

The ADC Driver does not handle clock setup (from PCC) or any kind of pin configuration (done by PORT module). This is handled by the Clock Manager and PORT module, respectively. The driver assumes that correct clock configurations have been made, so it is the user's responsibility to set up clocking and pin configurations correctly.

### Triggering a conversion

There are two separate ways for triggering an ADC conversion from a control channel:

1. Software triggering Only conversion from first control channel may be triggered from software - must enabled at converter configuration Initiated by writing a valid input channel ID to the first control channel - use [ADC\\_Drv\\_ConfigChan\(\)](#).
2. Hardware triggering Conversion from any control channel may be hardware triggered - however for first control channel it must be enabled at converter configuration.

### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
* ${S32SDK_PATH}\platform\drivers\src\adc_driver.c
*
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
* ${S32SDK_PATH}\platform\drivers\inc\
*
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

[Clock Manager](#)

### Data Structures

- struct [adc\\_converter\\_config\\_t](#)  
*Defines the converter configuration. [More...](#)*
- struct [adc\\_compare\\_config\\_t](#)  
*Defines the hardware compare configuration. [More...](#)*
- struct [adc\\_average\\_config\\_t](#)  
*Defines the hardware average configuration. [More...](#)*
- struct [adc\\_chan\\_config\\_t](#)  
*Defines the control channel configuration. [More...](#)*
- struct [adc\\_calibration\\_t](#)  
*Defines the user calibration configuration. [More...](#)*

### Enumerations

- enum [adc\\_clk\\_divide\\_t](#) { [ADC\\_CLK\\_DIVIDE\\_1](#) = 0x00U, [ADC\\_CLK\\_DIVIDE\\_2](#) = 0x01U, [ADC\\_CLK\\_DIVIDE\\_4](#) = 0x02U, [ADC\\_CLK\\_DIVIDE\\_8](#) = 0x03U }

*Clock Divider selection.*

- enum `adc_resolution_t` { `ADC_RESOLUTION_8BIT` = 0x00U, `ADC_RESOLUTION_12BIT` = 0x01U, `ADC_RESOLUTION_10BIT` = 0x02U }

*Conversion resolution selection.*

- enum `adc_input_clock_t` { `ADC_CLK_ALT_1` = 0x00U, `ADC_CLK_ALT_2` = 0x01U, `ADC_CLK_ALT_3` = 0x02U, `ADC_CLK_ALT_4` = 0x03U }

*Input clock source selection.*

- enum `adc_trigger_t` { `ADC_TRIGGER_SOFTWARE` = 0x00U, `ADC_TRIGGER_HARDWARE` = 0x01U }

*Trigger type selection.*

- enum `adc_pretrigger_sel_t` { `ADC_PRETRIGGER_SEL_PDB` = 0x00U, `ADC_PRETRIGGER_SEL_TRGMUX` = 0x01U, `ADC_PRETRIGGER_SEL_SW` = 0x02U }

*Pretrigger types selectable from Trigger Latching and Arbitration Unit.*

- enum `adc_trigger_sel_t` { `ADC_TRIGGER_SEL_PDB` = 0x00U, `ADC_TRIGGER_SEL_TRGMUX` = 0x01U }

*Trigger source selectable from Trigger Latching and Arbitration Unit.*

- enum `adc_sw_pretrigger_t` { `ADC_SW_PRETRIGGER_DISABLED` = 0x00U, `ADC_SW_PRETRIGGER_0` = 0x04U, `ADC_SW_PRETRIGGER_1` = 0x05U, `ADC_SW_PRETRIGGER_2` = 0x06U, `ADC_SW_PRETRIGGER_3` = 0x07U }

*Software pretriggers which may be set from Trigger Latching and Arbitration Unit.*

- enum `adc_voltage_reference_t` { `ADC_VOLTAGEREF_VREF` = 0x00U, `ADC_VOLTAGEREF_VALT` = 0x01U }

*Voltage reference selection.*

- enum `adc_average_t` { `ADC_AVERAGE_4` = 0x00U, `ADC_AVERAGE_8` = 0x01U, `ADC_AVERAGE_16` = 0x02U, `ADC_AVERAGE_32` = 0x03U }

*Hardware average selection.*

- enum `adc_inputchannel_t` { `ADC_INPUTCHAN_EXT0` = 0x00U, `ADC_INPUTCHAN_EXT1` = 0x01U, `ADC_INPUTCHAN_EXT3` = 0x03U, `ADC_INPUTCHAN_EXT4` = 0x04U, `ADC_INPUTCHAN_EXT5` = 0x05U, `ADC_INPUTCHAN_EXT6` = 0x06U, `ADC_INPUTCHAN_EXT7` = 0x07U, `ADC_INPUTCHAN_EXT9` = 0x09U, `ADC_INPUTCHAN_EXT10` = 0x0AU, `ADC_INPUTCHAN_EXT11` = 0x0BU, `ADC_INPUTCHAN_EXT12` = 0x0CU, `ADC_INPUTCHAN_EXT13` = 0x0DU, `ADC_INPUTCHAN_EXT14` = 0x0EU, `ADC_INPUTCHAN_DISABLED` = `ADC_SC1_ADCH_MASK`, `ADC_INPUTCHAN_INT0` = 0x15, `ADC_INPUTCHAN_INT1` = 0x16, `ADC_INPUTCHAN_INT2` = 0x17, `ADC_INPUTCHAN_INT3` = 0x1C, `ADC_INPUTCHAN_TEMP` = 0x1A, `ADC_INPUTCHAN_BANDGAP` = 0x1B, `ADC_INPUTCHAN_VREFSH` = 0x1D, `ADC_INPUTCHAN_VREFSL` = 0x1E, `ADC_INPUTCHAN_SUPPLY_VDD` = 0xF00U, `ADC_INPUTCHAN_SUPPLY_VDDA` = 0xF01U, `ADC_INPUTCHAN_SUPPLY_VREFH` = 0xF02U, `ADC_INPUTCHAN_SUPPLY_VDD_3V` = 0xF03U, `ADC_INPUTCHAN_SUPPLY_VDD_FLASH_3V` = 0xF04U, `ADC_INPUTCHAN_SUPPLY_VDD_LV` = 0xF05U }

*Enumeration of input channels assignable to a control channel.*

**Note 0:** entries in this enum are affected by `::FEATURE_ADC_NUM_EXT_CHANS`, which is device dependent and controlled from "device\_name" features.h file.

- enum `adc_latch_clear_t` { `ADC_LATCH_CLEAR_WAIT`, `ADC_LATCH_CLEAR_FORCE` }

*Defines the trigger latch clear method Implements : `adc_latch_clear_t` Class.*

## Converter

Converter specific methods. These are used to configure and use the A/D Converter specific functionality, including:

- clock input and divider
- sample time in A/D clocks

- resolution
- trigger source
- voltage reference
- enable DMA
- enable continuous conversion on one channel

To start a conversion, a control channel (see [Channel Configuration](#)) and a trigger source must be configured. Once a conversion is started, the user application can wait for it to be finished by calling the [ADC\\_DRV\\_WaitConvDone\(\)](#) function.

Only the first control channel can be triggered by software. To start a conversion in this case, an input channel must be written in the channel selection register using the [ADC\\_DRV\\_ConfigChan\(\)](#) method. Writing a value to the control channel while a conversion is being performed on that channel will start a new conversion.

- void [ADC\\_DRV\\_InitConverterStruct](#) ([adc\\_converter\\_config\\_t](#) \*const config)  
*Initializes the converter configuration structure.*
- void [ADC\\_DRV\\_ConfigConverter](#) (const uint32\_t instance, const [adc\\_converter\\_config\\_t](#) \*const config)  
*Configures the converter with the given configuration structure.*
- void [ADC\\_DRV\\_GetConverterConfig](#) (const uint32\_t instance, [adc\\_converter\\_config\\_t](#) \*const config)  
*Gets the current converter configuration.*
- void [ADC\\_DRV\\_Reset](#) (const uint32\_t instance)  
*Resets the converter (sets all configurations to reset values)*
- void [ADC\\_DRV\\_WaitConvDone](#) (const uint32\_t instance)  
*Waits for a conversion/calibration to finish.*
- bool [ADC\\_DRV\\_GetConvCompleteFlag](#) (const uint32\_t instance, const uint8\_t chanIndex)  
*Gets the control channel Conversion Complete Flag state.*

## Hardware Compare

The Hardware Compare feature of the S32K144 ADC is a versatile mechanism that can be used to monitor that a value is within certain values. Measurements can be monitored to be within certain ranges:

- less than/ greater than a fixed value
- inside or outside of a certain range

Two compare values can be configured (the second value is used only for range function mode). The compare values must be written in 12-bit resolution mode regardless of the actual used resolution mode.

Once the hardware compare feature is enabled, a conversion is considered complete only when the measured value is within the allowable range set by the configuration.

- void [ADC\\_DRV\\_InitHwCompareStruct](#) ([adc\\_compare\\_config\\_t](#) \*const config)  
*Initializes the Hardware Compare configuration structure.*
- void [ADC\\_DRV\\_ConfigHwCompare](#) (const uint32\_t instance, const [adc\\_compare\\_config\\_t](#) \*const config)  
*Configures the Hardware Compare feature with the given configuration structure.*
- void [ADC\\_DRV\\_GetHwCompareConfig](#) (const uint32\_t instance, [adc\\_compare\\_config\\_t](#) \*const config)  
*Gets the current Hardware Compare configuration.*

## Hardware Average

The Hardware Average feature of the S32K144 allows for a set of measurements to be averaged together as a single conversion. The number of samples to be averaged is selectable (4, 8, 16 or 32 samples).

- void [ADC\\_DRV\\_InitHwAverageStruct](#) ([adc\\_average\\_config\\_t](#) \*const config)  
*Initializes the Hardware Average configuration structure.*
- void [ADC\\_DRV\\_ConfigHwAverage](#) (const uint32\_t instance, const [adc\\_average\\_config\\_t](#) \*const config)  
*Configures the Hardware Average feature with the given configuration structure.*
- void [ADC\\_DRV\\_GetHwAverageConfig](#) (const uint32\_t instance, [adc\\_average\\_config\\_t](#) \*const config)  
*Gets the current Hardware Average configuration.*

## Channel configuration

Control register specific functions. These functions control configurations for each control channel (input channel selection and interrupt enable).

When software triggering is enabled, calling the [ADC\\_DRV\\_ConfigChan\(\)](#) method for control channel 0 starts a new conversion.

After a conversion is finished, the result can be retrieved using the [ADC\\_DRV\\_GetChanResult\(\)](#) method.

- void [ADC\\_DRV\\_InitChanStruct](#) ([adc\\_chan\\_config\\_t](#) \*const config)  
*Initializes the control channel configuration structure.*
- void [ADC\\_DRV\\_ConfigChan](#) (const uint32\_t instance, const uint8\_t chanIndex, const [adc\\_chan\\_config\\_t](#) \*const config)  
*Configures the selected control channel with the given configuration structure.*
- void [ADC\\_DRV\\_GetChanConfig](#) (const uint32\_t instance, const uint8\_t chanIndex, [adc\\_chan\\_config\\_t](#) \*const config)  
*Gets the current control channel configuration for the selected channel index.*
- void [ADC\\_DRV\\_SetSwPretrigger](#) (const uint32\_t instance, const [adc\\_sw\\_pretrigger\\_t](#) swPretrigger)  
*This function sets the software pretrigger - affects only first 4 control channels.*
- void [ADC\\_DRV\\_GetChanResult](#) (const uint32\_t instance, const uint8\_t chanIndex, uint16\_t \*const result)  
*Gets the last result for the selected control channel.*

## Automatic Calibration

These methods control the Calibration feature of the ADC.

The [ADC\\_DRV\\_AutoCalibration\(\)](#) method can be called to execute a calibration sequence, or a calibration can be retrieved with the [ADC\\_DRV\\_GetUserCalibration\(\)](#) and saved to non-volatile storage, to avoid calibration on every power-on. The calibration structure can be written with the [ADC\\_DRV\\_ConfigUserCalibration\(\)](#) method.

- void [ADC\\_DRV\\_AutoCalibration](#) (const uint32\_t instance)  
*Executes an Auto-Calibration.*
- void [ADC\\_DRV\\_InitUserCalibrationStruct](#) ([adc\\_calibration\\_t](#) \*const config)  
*Initializes the User Calibration configuration structure.*
- void [ADC\\_DRV\\_ConfigUserCalibration](#) (const uint32\_t instance, const [adc\\_calibration\\_t](#) \*const config)  
*Configures the User Calibration feature with the given configuration structure.*
- void [ADC\\_DRV\\_GetUserCalibration](#) (const uint32\_t instance, [adc\\_calibration\\_t](#) \*const config)  
*Gets the current User Calibration configuration.*

## Interrupts

This method returns the interrupt number for an ADC instance, which can be used to configure the interrupt, like in Interrupt Manager.

- IRQn\_Type [ADC\\_DRV\\_GetInterruptNumber](#) (const uint32\_t instance)  
*Returns the interrupt number for the ADC instance.*

## Latched triggers processing

These functions provide basic operations for using the trigger latch mechanism.

- void [ADC\\_DRV\\_ClearLatchedTriggers](#) (const uint32\_t instance, const [adc\\_latch\\_clear\\_t](#) clearMode)  
*Clear latched triggers under processing.*
- void [ADC\\_DRV\\_ClearTriggerErrors](#) (const uint32\_t instance)  
*Clear all latch trigger error.*
- uint32\_t [ADC\\_DRV\\_GetTriggerErrorFlags](#) (const uint32\_t instance)  
*Get the trigger error flags bits of the ADC instance.*

## 16.18.2 Data Structure Documentation

### 16.18.2.1 struct adc\_converter\_config\_t

Defines the converter configuration.

This structure is used to configure the ADC converter

Implements : [adc\\_converter\\_config\\_t\\_Class](#)

Definition at line 249 of file [adc\\_driver.h](#).

#### Data Fields

- [adc\\_clk\\_divide\\_t](#) clockDivide
- uint8\_t sampleTime
- [adc\\_resolution\\_t](#) resolution
- [adc\\_input\\_clock\\_t](#) inputClock
- [adc\\_trigger\\_t](#) trigger
- [adc\\_pretrigger\\_sel\\_t](#) pretriggerSel
- [adc\\_trigger\\_sel\\_t](#) triggerSel
- bool dmaEnable
- [adc\\_voltage\\_reference\\_t](#) voltageRef
- bool continuousConvEnable
- bool supplyMonitoringEnable

#### Field Documentation

##### 16.18.2.1.1 [adc\\_clk\\_divide\\_t](#) clockDivide

Divider of the input clock for the ADC

Definition at line 251 of file [adc\\_driver.h](#).

##### 16.18.2.1.2 bool continuousConvEnable

Enable Continuous conversions

Definition at line 260 of file [adc\\_driver.h](#).

**16.18.2.1.3 bool dmaEnable**

Enable DMA for the ADC

Definition at line 258 of file `adc_driver.h`.

**16.18.2.1.4 adc\_input\_clock\_t inputClock**

Input clock source

Definition at line 254 of file `adc_driver.h`.

**16.18.2.1.5 adc\_pretrigger\_sel\_t pretriggerSel**

Pretrigger source selected from Trigger Latching and Arbitration Unit - affects only the first 4 control channels

Definition at line 256 of file `adc_driver.h`.

**16.18.2.1.6 adc\_resolution\_t resolution**

ADC resolution (8,10,12 bit)

Definition at line 253 of file `adc_driver.h`.

**16.18.2.1.7 uint8\_t sampleTime**

Sample time in AD Clocks

Definition at line 252 of file `adc_driver.h`.

**16.18.2.1.8 bool supplyMonitoringEnable**

Only available for ADC 0. Enable internal supply monitoring - enables measurement of ADC\_INPUTCHAN\_SUPPLY sources.

Definition at line 261 of file `adc_driver.h`.

**16.18.2.1.9 adc\_trigger\_t trigger**

ADC trigger type (software, hardware) - affects only the first control channel

Definition at line 255 of file `adc_driver.h`.

**16.18.2.1.10 adc\_trigger\_sel\_t triggerSel**

Trigger source selected from Trigger Latching and Arbitration Unit

Definition at line 257 of file `adc_driver.h`.

**16.18.2.1.11 adc\_voltage\_reference\_t voltageRef**

Voltage reference used

Definition at line 259 of file `adc_driver.h`.

**16.18.2.2 struct adc\_compare\_config\_t**

Defines the hardware compare configuration.

This structure is used to configure the hardware compare feature for the ADC

Implements : `adc_compare_config_t_Class`

Definition at line 272 of file `adc_driver.h`.

**Data Fields**

- bool `compareEnable`



- bool [compareGreaterThanEnable](#)
- bool [compareRangeFuncEnable](#)
- uint16\_t [compVal1](#)
- uint16\_t [compVal2](#)

#### Field Documentation

##### 16.18.2.2.1 bool compareEnable

Enable the compare feature

Definition at line 274 of file `adc_driver.h`.

##### 16.18.2.2.2 bool compareGreaterThanEnable

Enable Greater-Than functionality

Definition at line 275 of file `adc_driver.h`.

##### 16.18.2.2.3 bool compareRangeFuncEnable

Enable Range functionality

Definition at line 276 of file `adc_driver.h`.

##### 16.18.2.2.4 uint16\_t compVal1

First Compare Value

Definition at line 277 of file `adc_driver.h`.

##### 16.18.2.2.5 uint16\_t compVal2

Second Compare Value

Definition at line 278 of file `adc_driver.h`.

##### 16.18.2.3 struct adc\_average\_config\_t

Defines the hardware average configuration.

This structure is used to configure the hardware average feature for the ADC

Implements : `adc_average_config_t_Class`

Definition at line 289 of file `adc_driver.h`.

#### Data Fields

- bool [hwAvgEnable](#)
- [adc\\_average\\_t](#) [hwAverage](#)

#### Field Documentation

##### 16.18.2.3.1 adc\_average\_t hwAverage

Selection for number of samples used for averaging

Definition at line 292 of file `adc_driver.h`.

##### 16.18.2.3.2 bool hwAvgEnable

Enable averaging functionality

Definition at line 291 of file `adc_driver.h`.

#### 16.18.2.4 struct adc\_chan\_config\_t

Defines the control channel configuration.

This structure is used to configure a control channel of the ADC

Implements : `adc_chan_config_t_Class`

Definition at line 303 of file `adc_driver.h`.

##### Data Fields

- bool `interruptEnable`
- `adc_inputchannel_t` `channel`

##### Field Documentation

##### 16.18.2.4.1 adc\_inputchannel\_t channel

Selection of input channel for measurement

Definition at line 306 of file `adc_driver.h`.

##### 16.18.2.4.2 bool interruptEnable

Enable interrupts for this channel

Definition at line 305 of file `adc_driver.h`.

#### 16.18.2.5 struct adc\_calibration\_t

Defines the user calibration configuration.

This structure is used to configure the user calibration parameters of the ADC.

Implements : `adc_calibration_t_Class`

Definition at line 317 of file `adc_driver.h`.

##### Data Fields

- uint16\_t `userGain`
- uint16\_t `userOffset`

##### Field Documentation

##### 16.18.2.5.1 uint16\_t userGain

User-configurable gain

Definition at line 319 of file `adc_driver.h`.

##### 16.18.2.5.2 uint16\_t userOffset

User-configurable Offset (2's complement, subtracted from result)

Definition at line 320 of file `adc_driver.h`.

#### 16.18.3 Enumeration Type Documentation

##### 16.18.3.1 enum adc\_average\_t

Hardware average selection.

Implements : `adc_average_t_Class`

## Enumerator

- ADC\_AVERAGE\_4** Hardware average of 4 samples.
- ADC\_AVERAGE\_8** Hardware average of 8 samples.
- ADC\_AVERAGE\_16** Hardware average of 16 samples.
- ADC\_AVERAGE\_32** Hardware average of 32 samples.

Definition at line 154 of file `adc_driver.h`.

16.18.3.2 enum `adc_clk_divide_t`

Clock Divider selection.

Implements : `adc_clk_divide_t_Class`

## Enumerator

- ADC\_CLK\_DIVIDE\_1** Input clock divided by 1.
- ADC\_CLK\_DIVIDE\_2** Input clock divided by 2.
- ADC\_CLK\_DIVIDE\_4** Input clock divided by 4.
- ADC\_CLK\_DIVIDE\_8** Input clock divided by 8.

Definition at line 57 of file `adc_driver.h`.

16.18.3.3 enum `adc_input_clock_t`

Input clock source selection.

Implements : `adc_input_clock_t_Class`

## Enumerator

- ADC\_CLK\_ALT\_1** Input clock alternative 1.
- ADC\_CLK\_ALT\_2** Input clock alternative 2.
- ADC\_CLK\_ALT\_3** Input clock alternative 3.
- ADC\_CLK\_ALT\_4** Input clock alternative 4.

Definition at line 82 of file `adc_driver.h`.

16.18.3.4 enum `adc_inputchannel_t`

Enumeration of input channels assignable to a control channel.

**Note 0:** entries in this enum are affected by `::FEATURE_ADC_NUM_EXT_CHANS`, which is device dependent and controlled from `"device_name"_features.h` file.

**Note 1:** the actual number of external channels may differ between device packages and ADC instances. Reading a channel that is not connected externally, will return a random value within the range. Please refer to the Reference Manual for the maximum number of external channels for each device variant and ADC instance.

**Note 2:** `ADC_INPUTCHAN_SUPPLY_` select which internal supply channel to be measured. They are only available for ADC0 and measured internally via internal input channel 0. Please note that supply monitoring needs to be enabled separately via dedicated flag in [adc\\_converter\\_config\\_t](#).

Implements : `adc_inputchannel_t_Class`

## Enumerator

- ADC\_INPUTCHAN\_EXT0** External input channel 0
- ADC\_INPUTCHAN\_EXT1** External input channel 1

**ADC\_INPUTCHAN\_EXT3** External input channel 3  
**ADC\_INPUTCHAN\_EXT4** External input channel 4  
**ADC\_INPUTCHAN\_EXT5** External input channel 5  
**ADC\_INPUTCHAN\_EXT6** External input channel 6  
**ADC\_INPUTCHAN\_EXT7** External input channel 7  
**ADC\_INPUTCHAN\_EXT9** External input channel 9  
**ADC\_INPUTCHAN\_EXT10** External input channel 10  
**ADC\_INPUTCHAN\_EXT11** External input channel 11  
**ADC\_INPUTCHAN\_EXT12** External input channel 12  
**ADC\_INPUTCHAN\_EXT13** External input channel 13  
**ADC\_INPUTCHAN\_EXT14** External input channel 14  
**ADC\_INPUTCHAN\_DISABLED** Channel disabled  
**ADC\_INPUTCHAN\_INT0** Internal input channel 0  
**ADC\_INPUTCHAN\_INT1** Internal input channel 1  
**ADC\_INPUTCHAN\_INT2** Internal input channel 2  
**ADC\_INPUTCHAN\_INT3** Internal input channel 3  
**ADC\_INPUTCHAN\_TEMP** Temperature Sensor  
**ADC\_INPUTCHAN\_BANDGAP** Band Gap  
**ADC\_INPUTCHAN\_VREFSH** Voltage Reference Select High  
**ADC\_INPUTCHAN\_VREFSL** Voltage Reference Select Low  
**ADC\_INPUTCHAN\_SUPPLY\_VDD** Monitor internal supply 5 V input VDD supply.  
**ADC\_INPUTCHAN\_SUPPLY\_VDDA** Monitor internal supply 5 V input analog supply.  
**ADC\_INPUTCHAN\_SUPPLY\_VREFH** Monitor internal supply ADC reference supply.  
**ADC\_INPUTCHAN\_SUPPLY\_VDD\_3V** Monitor internal supply 3.3 V oscillator regulator output.  
**ADC\_INPUTCHAN\_SUPPLY\_VDD\_FLASH\_3V** Monitor internal supply 3.3 V flash regulator output.  
**ADC\_INPUTCHAN\_SUPPLY\_VDD\_LV** Monitor internal supply 1.2 V core regulator output.

Definition at line 177 of file adc\_driver.h.

#### 16.18.3.5 enum adc\_latch\_clear\_t

Defines the trigger latch clear method Implements : adc\_latch\_clear\_t\_Class.

##### Enumerator

**ADC\_LATCH\_CLEAR\_WAIT** Clear by waiting all latched triggers to be processed  
**ADC\_LATCH\_CLEAR\_FORCE** Process current trigger and clear all latched

Definition at line 327 of file adc\_driver.h.

#### 16.18.3.6 enum adc\_pretrigger\_sel\_t

Pretrigger types selectable from Trigger Latching and Arbitration Unit.

Implements : adc\_pretrigger\_sel\_t\_Class

##### Enumerator

**ADC\_PRETRIGGER\_SEL\_PDB** PDB pretrigger selected.  
**ADC\_PRETRIGGER\_SEL\_TRGMUX** TRGMUX pretrigger selected.  
**ADC\_PRETRIGGER\_SEL\_SW** Software pretrigger selected.

Definition at line 106 of file adc\_driver.h.

### 16.18.3.7 enum `adc_resolution_t`

Conversion resolution selection.

Implements : `adc_resolution_t_Class`

#### Enumerator

**`ADC_RESOLUTION_8BIT`** 8-bit resolution mode  
**`ADC_RESOLUTION_12BIT`** 12-bit resolution mode  
**`ADC_RESOLUTION_10BIT`** 10-bit resolution mode

Definition at line 70 of file `adc_driver.h`.

### 16.18.3.8 enum `adc_sw_pretrigger_t`

Software pretriggers which may be set from Trigger Latching and Arbitration Unit.

Implements : `adc_sw_pretrigger_t_Class`

#### Enumerator

**`ADC_SW_PRETRIGGER_DISABLED`** SW pretrigger disabled.  
**`ADC_SW_PRETRIGGER_0`** SW pretrigger 0.  
**`ADC_SW_PRETRIGGER_1`** SW pretrigger 1.  
**`ADC_SW_PRETRIGGER_2`** SW pretrigger 2.  
**`ADC_SW_PRETRIGGER_3`** SW pretrigger 3.

Definition at line 129 of file `adc_driver.h`.

### 16.18.3.9 enum `adc_trigger_sel_t`

Trigger source selectable from Trigger Latching and Arbitration Unit.

Implements : `adc_trigger_sel_t_Class`

#### Enumerator

**`ADC_TRIGGER_SEL_PDB`** PDB trigger selected.  
**`ADC_TRIGGER_SEL_TRGMUX`** TRGMUX trigger selected.

Definition at line 118 of file `adc_driver.h`.

### 16.18.3.10 enum `adc_trigger_t`

Trigger type selection.

Implements : `adc_trigger_t_Class`

#### Enumerator

**`ADC_TRIGGER_SOFTWARE`** Software trigger.  
**`ADC_TRIGGER_HARDWARE`** Hardware trigger.

Definition at line 95 of file `adc_driver.h`.

### 16.18.3.11 enum `adc_voltage_reference_t`

Voltage reference selection.

Implements : `adc_voltage_reference_t_Class`

## Enumerator

**ADC\_VOLTAGEREF\_VREF** VrefH and VrefL as Voltage reference.

**ADC\_VOLTAGEREF\_VALT** ValtH and ValtL as Voltage reference.

Definition at line 143 of file adc\_driver.h.

## 16.18.4 Function Documentation

## 16.18.4.1 void ADC\_DRV\_AutoCalibration ( const uint32\_t instance )

Executes an Auto-Calibration.

This functions executes an Auto-Calibration sequence. It is recommended to run this sequence before using the ADC converter.

## Parameters

in	instance	instance number
----	----------	-----------------

Definition at line 555 of file adc\_driver.c.

## 16.18.4.2 void ADC\_DRV\_ClearLatchedTriggers ( const uint32\_t instance, const adc\_latch\_clear\_t clearMode )

Clear latched triggers under processing.

This function clears all trigger latched flags of the ADC instance. This function must be called after the hardware trigger source for the ADC has been deactivated.

## Parameters

in	instance	instance number of the ADC
in	clearMode	The clearing method for the latched triggers <ul style="list-style-type: none"> <li>• ADC_LATCH_CLEAR_WAIT : Wait for all latched triggers to be processed.</li> <li>• ADC_LATCH_CLEAR_FORCE : Clear latched triggers and wait for trigger being process to finish.</li> </ul>

Definition at line 712 of file adc\_driver.c.

## 16.18.4.3 void ADC\_DRV\_ClearTriggerErrors ( const uint32\_t instance )

Clear all latch trigger error.

This function clears all trigger error flags of the ADC instance.

## Parameters

in	instance	instance number of the ADC
----	----------	----------------------------

Definition at line 737 of file adc\_driver.c.

## 16.18.4.4 void ADC\_DRV\_ConfigChan ( const uint32\_t instance, const uint8\_t chanIndex, const adc\_chan\_config\_t \*const config )

Configures the selected control channel with the given configuration structure.

When Software Trigger mode is enabled, configuring control channel index 0, implicitly triggers a new conversion on the selected ADC input channel. Therefore, ADC\_DRV\_ConfigChan can be used for sw-triggering conversions.

Configuring any control channel while it is actively controlling a conversion (sw or hw triggered) will implicitly abort the on-going conversion.

**Parameters**

in	<i>instance</i>	instance number
in	<i>chanIndex</i>	the control channel index
in	<i>config</i>	the configuration structure

Definition at line 381 of file adc\_driver.c.

**16.18.4.5** void ADC\_DRV\_ConfigConverter ( const uint32\_t *instance*, const adc\_converter\_config\_t \*const *config* )

Configures the converter with the given configuration structure.

This function configures the ADC converter with the options provided in the provided structure.

**Parameters**

in	<i>instance</i>	instance number
in	<i>config</i>	the configuration structure

Definition at line 94 of file adc\_driver.c.

**16.18.4.6** void ADC\_DRV\_ConfigHwAverage ( const uint32\_t *instance*, const adc\_average\_config\_t \*const *config* )

Configures the Hardware Average feature with the given configuration structure.

This function sets the configuration for the Hardware Average feature.

**Parameters**

in	<i>instance</i>	instance number
in	<i>config</i>	the configuration structure

Definition at line 318 of file adc\_driver.c.

**16.18.4.7** void ADC\_DRV\_ConfigHwCompare ( const uint32\_t *instance*, const adc\_compare\_config\_t \*const *config* )

Configures the Hardware Compare feature with the given configuration structure.

This functions sets the configuration for the Hardware Compare feature using the configuration structure.

**Parameters**

in	<i>instance</i>	instance number
in	<i>config</i>	the configuration structure

Definition at line 255 of file adc\_driver.c.

**16.18.4.8** void ADC\_DRV\_ConfigUserCalibration ( const uint32\_t *instance*, const adc\_calibration\_t \*const *config* )

Configures the User Calibration feature with the given configuration structure.

This function sets the configuration for the user calibration registers.

**Parameters**

in	<i>instance</i>	instance number
in	<i>config</i>	the configuration structure

Definition at line 658 of file adc\_driver.c.

**16.18.4.9** void ADC\_DRV\_GetChanConfig ( const uint32\_t *instance*, const uint8\_t *chanIndex*, adc\_chan\_config\_t \*const *config* )

Gets the current control channel configuration for the selected channel index.

This function returns the configuration for a control channel

**Parameters**

in	<i>instance</i>	instance number
in	<i>chanIndex</i>	the control channel index
out	<i>config</i>	the configuration structure

Definition at line 406 of file adc\_driver.c.

**16.18.4.10** void ADC\_DRV\_GetChanResult ( const uint32\_t *instance*, const uint8\_t *chanIndex*, uint16\_t \*const *result* )

Gets the last result for the selected control channel.

This function returns the conversion result from a control channel.

**Parameters**

in	<i>instance</i>	instance number
in	<i>chanIndex</i>	the converter control channel index
out	<i>result</i>	the result raw value

Definition at line 517 of file adc\_driver.c.

**16.18.4.11** bool ADC\_DRV\_GetConvCompleteFlag ( const uint32\_t *instance*, const uint8\_t *chanIndex* )

Gets the control channel Conversion Complete Flag state.

This function returns the state of the Conversion Complete flag for a control channel. This flag is set when a conversion is complete or the condition generated by the Hardware Compare feature is evaluated to true.

**Parameters**

in	<i>instance</i>	instance number
in	<i>chanIndex</i>	the adc control channel index

**Returns**

the Conversion Complete Flag state

Definition at line 490 of file adc\_driver.c.

**16.18.4.12** void ADC\_DRV\_GetConverterConfig ( const uint32\_t *instance*, adc\_converter\_config\_t \*const *config* )

Gets the current converter configuration.

This functions returns the configuration for converter in the form of a configuration structure.

**Parameters**

in	<i>instance</i>	instance number
out	<i>config</i>	the configuration structure

Definition at line 140 of file adc\_driver.c.

**16.18.4.13** void ADC\_DRV\_GetHwAverageConfig ( const uint32\_t *instance*, adc\_average\_config\_t \*const *config* )

Gets the current Hardware Average configuration.

This function returns the configuration for the Hardware Average feature.

**Parameters**

in	<i>instance</i>	instance number
----	-----------------	-----------------



out	<i>config</i>	the configuration structure
-----	---------------	-----------------------------

Definition at line 337 of file adc\_driver.c.

**16.18.4.14** void ADC\_DRV\_GetHwCompareConfig ( const uint32\_t *instance*, adc\_compare\_config\_t \*const *config* )

Gets the current Hardware Compare configuration.

This function returns the configuration for the Hardware Compare feature.

#### Parameters

in	<i>instance</i>	instance number
out	<i>config</i>	the configuration structure

Definition at line 277 of file adc\_driver.c.

**16.18.4.15** IRQn\_Type ADC\_DRV\_GetInterruptNumber ( const uint32\_t *instance* )

Returns the interrupt number for the ADC instance.

This function returns the interrupt number for the specified ADC instance.

#### Parameters

in	<i>instance</i>	instance number of the ADC
----	-----------------	----------------------------

#### Returns

irq\_number: the interrupt number (index) of the ADC instance, used to configure the interrupt

Definition at line 695 of file adc\_driver.c.

**16.18.4.16** uint32\_t ADC\_DRV\_GetTriggerErrorFlags ( const uint32\_t *instance* )

Get the trigger error flags bits of the ADC instance.

This function returns the trigger error flags bits of the ADC instance.

#### Parameters

in	<i>instance</i>	instance number of the ADC
----	-----------------	----------------------------

#### Returns

trigErrorFlags The Trigger Error Flags bit-mask

Definition at line 753 of file adc\_driver.c.

**16.18.4.17** void ADC\_DRV\_GetUserCalibration ( const uint32\_t *instance*, adc\_calibration\_t \*const *config* )

Gets the current User Calibration configuration.

This function returns the current user calibration register values.

#### Parameters

in	<i>instance</i>	instance number
out	<i>config</i>	the configuration structure

Definition at line 677 of file adc\_driver.c.

**16.18.4.18** void ADC\_DRV\_InitChanStruct ( adc\_chan\_config\_t \*const *config* )

Initializes the control channel configuration structure.

This function initializes the control channel configuration structure to default values (Reference Manual resets). This function should be called on a structure before using it to configure a channel (ADC\_DRV\_ConfigChan), otherwise all members must be written by the caller. This function insures that all members are written with safe values, so the user can modify only the desired members.

#### Parameters

out	config	the configuration structure
-----	--------	-----------------------------

Definition at line 359 of file adc\_driver.c.

#### 16.18.4.19 void ADC\_DRV\_InitConverterStruct ( adc\_converter\_config\_t \*const config )

Initializes the converter configuration structure.

This function initializes the members of the [adc\\_converter\\_config\\_t](#) structure to default values (Reference Manual resets). This function should be called on a structure before using it to configure the converter with [ADC\\_DRV\\_InitConverterStruct\(\)](#), otherwise all members must be written (initialized) by the user. This function insures that all members are written with safe values, so the user can modify only the desired members.

#### Parameters

out	config	the configuration structure
-----	--------	-----------------------------

Definition at line 69 of file adc\_driver.c.

#### 16.18.4.20 void ADC\_DRV\_InitHwAverageStruct ( adc\_average\_config\_t \*const config )

Initializes the Hardware Average configuration structure.

This function initializes the Hardware Average configuration structure to default values (Reference Manual resets). This function should be called before configuring the Hardware Average feature (ADC\_DRV\_ConfigHwAverage), otherwise all members must be written by the caller. This function insures that all members are written with safe values, so the user can modify the desired members.

#### Parameters

out	config	the configuration structure
-----	--------	-----------------------------

Definition at line 302 of file adc\_driver.c.

#### 16.18.4.21 void ADC\_DRV\_InitHwCompareStruct ( adc\_compare\_config\_t \*const config )

Initializes the Hardware Compare configuration structure.

This function initializes the Hardware Compare configuration structure to default values (Reference Manual resets). This function should be called before configuring the Hardware Compare feature (ADC\_DRV\_ConfigHwCompare), otherwise all members must be written by the caller. This function insures that all members are written with safe values, so the user can modify the desired members.

#### Parameters

out	config	the configuration structure
-----	--------	-----------------------------

Definition at line 236 of file adc\_driver.c.

#### 16.18.4.22 void ADC\_DRV\_InitUserCalibrationStruct ( adc\_calibration\_t \*const config )

Initializes the User Calibration configuration structure.

This function initializes the User Calibration configuration structure to default values (Reference Manual resets). This function should be called on a structure before using it to configure the User Calibration feature (ADC\_DRV\_InitUserCalibration), otherwise all members must be written by the caller. This function insures that all members are written with safe values, so the user can modify only the desired members. this function will check and reset clock divide based the adc frequency. an error will be displayed if frequency is greater than required clock for calibration.

**Parameters**

out	<i>config</i>	the configuration structure
-----	---------------	-----------------------------

Definition at line 642 of file adc\_driver.c.

**16.18.4.23** void ADC\_DRV\_Reset ( const uint32\_t *instance* )

Resets the converter (sets all configurations to reset values)

This function resets all the internal ADC registers to reset values.

**Parameters**

in	<i>instance</i>	instance number
----	-----------------	-----------------

Definition at line 178 of file adc\_driver.c.

**16.18.4.24** void ADC\_DRV\_SetSwPretrigger ( const uint32\_t *instance*, const adc\_sw\_pretrigger\_t *swPretrigger* )

This function sets the software pretrigger - affects only first 4 control channels.

**Parameters**

in	<i>instance</i>	instance number
in	<i>swPretrigger</i>	the swPretrigger to be enabled

Definition at line 426 of file adc\_driver.c.

**16.18.4.25** void ADC\_DRV\_WaitConvDone ( const uint32\_t *instance* )

Waits for a conversion/calibration to finish.

This functions waits for a conversion to complete by continuously polling the Conversion Active Flag.

**Parameters**

in	<i>instance</i>	instance number
----	-----------------	-----------------

Definition at line 470 of file adc\_driver.c.

## 16.19 Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL)

### 16.19.1 Detailed Description

Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL).

#### ADC PAL general consideration

The ADC PAL is an interface abstraction layer for multiple Analog to Digital Converter peripherals.

The ADC PAL allows configuration of groups of successive conversions started by a single trigger event.

Each conversion in a group is mapped to an ADC input channel - the **conversion group** is actually defined by an array of input channels, which is a member of the `adc_group_config_t` structure. The order of the input channels will also give the order of execution of the conversions within the group.

**Note:** all conversion groups need to be configured at PAL initialization time.

The trigger event for a group can be SW or HW, and needs to be selected at configuration time.

1. Execution of **SW triggered groups** may be started/stopped by calling a dedicated function `ADC_StartGroupConversion()`, `ADC_StopGroupConversion()`.
2. **HW triggered groups** need to be enabled for execution by calling a dedicated function - the actual execution will be started by the occurrence of the selected hardware trigger event `ADC_EnableHardwareTrigger()`, `ADC_DisableHardwareTrigger()`.

**Note:** for HW triggered groups the ADC PAL does not configure the peripherals which provide the triggering events (timers, counters, etc.) - they will need to be configured separately by the ADC PAL user.

Each group needs to have associated a **result buffer** which needs to be allocated by the PAL user. The length of the result buffer is defined by two configuration parameters:

\* `numChannels` - defines also the size of the `inputChannelArray`

\* `numSetsResultBuffer` - defines the number of sets of results which can be stored in the result buffer.

The *length of the result buffer* = `numChannels * numSetsResultBuffer`. Each time a group of conversions finishes execution, a set of results for all conversions in the group will be copied by the PAL into the corresponding result buffer. The PAL considers the result buffer as circular, with the length configured via previously described.

On some platforms, HW triggered groups may support **delay(s)** between the occurrence of the HW trigger event and the actual start of conversions. This feature can be controlled for each HW triggered group via `delayType` and `delayArray` parameters in `adc_group_config_t`. For SW triggered groups, these parameters are ignored. For details please refer to ADC PAL platform specific information.

Each group can also have associated a **notification callback** which will be executed when all conversions finish execution. The callback shall receive as parameter a pointer to `adc_callback_info_t` containing the *group index* for which the notification is called, and *result buffer tail* - offset of the most recent conversion result in the result buffer. Notifications can be enabled and disabled using `ADC_EnableNotification()` and `ADC_DisableNotification()`. By default the notification is set to active when enabling a HW triggered group or starting a SW triggered group.

**Note:** The notification callback may be set to NULL and thus it will not be called.

For SW triggered groups, **continuous mode** can be enabled at configuration time.

E.g.: a group with 3 conversions `InputCh0`, `InputCh1`, `InputCh2` -> with continuous mode enabled will continuously repeat the series of conversions until it is stopped: `InputCh0`, `InputCh1`, `InputCh2`, `InputCh0`, `InputCh1`, `InputCh2`,...

The user needs to dimension accordingly the result buffer, such that it has sufficient time to read the results before they are overwritten.

For HW triggered groups, continuous mode parameter is not available.

The ADC PAL implicitly configures and uses other peripherals besides ADC - these resources should not be used simultaneously from other parts of the application. For details please refer to the platform specific details.

The ADC PAL module needs to include a configuration file named `adc_pal_cfg.h`, which defines which IPs are used.

The ADC PAL allows configuration of platform specific parameters via a pointer to a platform specific structure, following the naming convention: `extension_adc_<platform>_t`. E.g.: `extension_adc_s32k1xx_t`

**Important note**

The ADC PAL configuration structure passed via reference to [ADC\\_Init\(\)](#), including all arrays referenced by structure members, must be persistent throughout the usage of the ADC PAL. Storing them to memory sections which get freed or altered during ADC PAL usage, will lead to unpredictable behavior.

**Platform specific details****S32K1xx device family**

On these platforms, each instance of ADC PAL uses:

- one instance of PDB linked to the selected ADC (ADCn - PDBn) - used for both SW and HW triggered groups
- the TRGMUX\_TARGET\_MODULE\_PDBn\_TRG\_IN targets from TRGMUX - used only for HW triggered groups

These platforms are supported by the ADC PAL of type **ADC\_PAL\_S32K1xx**.

**Important details:**

1. The PAL supports configuring any number of conversion groups at PAL initialization time, but every time a HW/SW triggered group is enabled/started, the underlying hardware peripherals are reconfigured.
2. The same input channel may appear multiple times in a group.

**Group delay support:**

- no delay between HW trigger event and conversions start:  
*delayType* = ADC\_DELAY\_TYPE\_NO\_DELAY and *delayArray* = NULL
- group delay between HW trigger event and the start of the first conversion in the group - the rest of conversions start right after the previous one  
*delayType* = ADC\_DELAY\_TYPE\_GROUP\_DELAY and *delayArray* set to point to a single uint16\_t variable storing the delay value, expressed in PDB ticks (affected by PDB prescaler configurable via config extension)
- individual delays between HW trigger event and the start of each conversion in the group *delayType* = ADC↔\_DELAY\_TYPE\_INDIVIDUAL\_DELAY and *delayArray* set to point to an uin16\_t array with length equal with the number of conversions in the group  
Delays are expressed in PDB ticks (affected by PDB prescaler configurable via config extension). Delay values are measured relative to the trigger event. When a delay expires, a PDB pretrigger is issued.  
**Note:** the pretriggers must not occur while another conversion in the group is running, otherwise the ADC freezes. It is the user's responsibility to make sure they do not overlap, i.e. *delayN\_plus\_1* > (*delayN* + *conversion\_duration*).

**MPC5746C and MPC5748G device families**

On these platforms, each instance of ADC PAL uses:

- one instance of BCTU - used only for HW triggered groups
- all ADC instances connected to the selected BCTU instance. Please note that the ADC instances may have different resolutions

These platforms are supported by the ADC PAL of type **ADC\_PAL\_MPC574xC\_G\_R**.

**Group delay support:**

- groups do not support delays, so in `adc_group_config_t` structures `delayType` must be set to `ADC_DELAY_TYPE_NO_DELAY` and `delayArray` to `NULL`, in `adc_group_config_t`.

Important details:

- The PAL supports any number of **SW triggered** conversion groups at PAL initialization time. SW triggered groups will be configured directly in ADC, each time `ADC_StartGroupConversion()` is called.
- The maximum supported number of **HW triggered** conversion groups is expressed in two steps:
  - for groups which include a minimum of 2 conversions: the total number of conversions within all these groups shall be less than or equal with the number of BCTU LIST HW registers. (E.g. 1 group of 8 conversions & 1 group of 24 conversions:  $8 + 24 \leq 32$ )
  - for groups which include a single conversion: the total number of such groups shall be less than or equal with the total number of BCTU Triggers minus the number of configured groups with at least 2 conversions
- An input channel may only appear once in the group, otherwise the last conversion result will appear for each occurrence of the channel index in the group. This is a platform limitation: BCTU has only a single result register per ADC instance, and the ADC has a single result register per channel.
- A conversion group (SW and HW triggered) can target only conversions on a single ADC instance.
- The same trigger source cannot be assigned to multiple HW triggered groups.
- Multiple HW triggered groups may be enabled simultaneously.  
However, the user must make sure that the actual HW trigger events do not occur simultaneously and that conversions from multiple groups do not overlap in time. Otherwise hardware errors may occur and results may be overwritten.

#### MPC574xP and S32Rx7x device families

On these platforms, each instance of ADC PAL uses:

- one instance of CTU - used only for HW triggered groups and statically configured to CTU triggered mode
- all ADC instances connected to the selected CTU instance

These platforms are supported by the ADC PAL of type **ADC\_PAL\_SAR\_CTU**.

Group delay support:

- no delay between HW trigger event and conversions start:  
`delayType = ADC_DELAY_TYPE_NO_DELAY` and `delayArray = NULL`
- group delay between HW trigger event and the start of the first conversion in the group - the rest of conversions start right after the previous one  
`delayType = ADC_DELAY_TYPE_GROUP_DELAY` and `delayArray` set to point to a single `uint16_t` variable storing the delay value, expressed in CTU ticks (affected by CTU prescaler)

Important details:

- The PAL supports any number of **SW triggered** conversion groups at PAL initialization time. SW triggered groups will be configured directly in ADC, each time `ADC_StartGroupConversion()` is called.
- The maximum supported number of **HW triggered** conversion groups is equal with the number of CTU result FIFOs - defined in platform header file as `CTU_FR_COUNT`. The total number of conversions in all HW triggered groups must be  $\leq$  the length of the CTU ADC command list - defined in platform header file as `CTU_CHANNEL_COUNT`.

3. A conversion group (SW and HW triggered) can target only conversions on a single ADC instance.
4. An input channel may only appear once in a SW triggered group, otherwise the last conversion result will appear for each occurrence of the channel index in the group. This is a platform limitation: the ADC has a single result register per channel. For HW triggered groups this restriction doesn't apply.
5. All HW triggered groups can be enabled simultaneously.  
However, the user must make sure that the actual HW trigger events do not occur simultaneously and that conversions from multiple groups do not overlap in time. Otherwise hardware errors may occur and results may be overwritten.
6. Each HW triggered group has assigned a CTU result FIFO. The number of channels in each group must be less than the CTU result FIFO length - note that not all FIFOs have the same length. FIFOs are assigned in the same order in which the HW triggered groups are configured in the PAL init state: FIFO#0 assigned to first group, FIFO#1 to second, etc.
7. The trigger sources enabled for a group can implicitly start also the rest of the enabled HW triggered groups. E.g. SourceX configured for group0, sourceY configured for group1. If both groups are enabled, when event from sourceX occurs, both group0 and group1 will execute; the same when event from sourceY occurs.

#### Integration guideline

##### Compilation units

The following files need to be compiled in the project:

```
{S32SDK_PATH}\platform\pal\src\adc\adc_pal.c
{S32SDK_PATH}\platform\pal\src\adc\adc_irq.c
```

Additionally, it is required to compile also the .c files from the dependencies listed for each ADC PAL type (please see Dependencies subsection below).

##### Include path

The following paths need to be added to the include path of the toolchain:

```
{S32SDK_PATH}\platform\pal\inc\
{S32SDK_PATH}\platform\drivers\inc\
```

An additional file, named *adc\_pal\_cfg.h*, must be created by the user and added to one of the include paths. The user has to add to the file the definitions of preprocessor symbols according to the ADC PAL type used. These symbols are specified in the next subsection.

When using the S32 SDK configuration tool, the file is generated by the configurator.

The pal type ADC\_PAL\_S32K1xx also requires:

```
{S32SDK_PATH}\platform\drivers\src\adc\
```

##### Compile symbols

1. Define for selecting one of the ADC PAL type to be used:

```
ADC_PAL_S32K1xx
ADC_PAL_MPC574xC_G_R
ADC_PAL_SAR_CTU
```

2. Define the maximum number of HW triggered groups which may be enabled simultaneously. For ADC\_PAL\_S32K1xx the maximum value of the define is 1.

```
ADC_PAL_MAX_NUM_HW_GROUPS_EN
```

3. For ADC\_PAL\_MPC574xC\_G\_R and ADC\_PAL\_SAR\_CTU types, define the total number of configured groups.

```
ADC_PAL_TOTAL_NUM_GROUPS
```

## Dependencies

[Interrupt Manager \(Interrupt\)](#)  
[OS Interface \(OSIF\)](#)

- The pal type `ADC_PAL_S32K1xx` also depends on:  
[ADC Driver](#)  
[PDB Driver](#)  
[TRGMUX Driver](#)
- The pal type `ADC_PAL_MPC574xC_G_R` also depends on:  
`adc_c55_driver`  
`bctu_driver`
- The pal type `ADC_PAL_SAR_CTU` also depends on:  
`adc_c55_driver`  
`ctu_driver`

## Data Structures

- struct [adc\\_group\\_config\\_t](#)  
*Defines the configuration structure for an ADC PAL conversion group. [More...](#)*
- struct [adc\\_config\\_t](#)  
*Defines the configuration structure for ADC PAL. [More...](#)*
- struct [extension\\_adc\\_s32k1xx\\_t](#)  
*Defines the extension structure for ADC S32K1xx. [More...](#)*

## Typedefs

- typedef [adc\\_inputchannel\\_t](#) [adc\\_input\\_chan\\_t](#)  
*Defines the enumeration with ADC PAL input channels.*
- typedef [trgmux\\_trigger\\_source\\_t](#) [adc\\_trigger\\_source\\_t](#)  
*Defines the enumeration with ADC PAL hardware trigger sources.*

## Enumerations

- enum [adc\\_delay\\_type\\_t](#) { `ADC_DELAY_TYPE_NO_DELAY` = 0u, `ADC_DELAY_TYPE_GROUP_DELAY` = 1u, `ADC_DELAY_TYPE_INDIVIDUAL_DELAY` = 2u }  
*Defines an enumeration which contains the types of delay configurations for ADC conversions within a group.*

## Functions

- status\_t [ADC\\_Init](#) (const [adc\\_instance\\_t](#) \*const instance, const [adc\\_config\\_t](#) \*const config)  
*Initializes the ADC PAL instance.*
- status\_t [ADC\\_Deinit](#) (const [adc\\_instance\\_t](#) \*const instance)  
*Deinitializes the ADC PAL instance.*
- status\_t [ADC\\_EnableHardwareTrigger](#) (const [adc\\_instance\\_t](#) \*const instance, const uint32\_t groupIdx)



*Enables the selected HW trigger for a conversion group, if the conversion group has support for HW trigger.*

- status\_t [ADC\\_DisableHardwareTrigger](#) (const [adc\\_instance\\_t](#) \*const instance, const uint32\_t groupIdx, const uint32\_t timeout)

*Disables the selected HW trigger for a conversion group, if the conversion group is HW triggered.*

- status\_t [ADC\\_StartGroupConversion](#) (const [adc\\_instance\\_t](#) \*const instance, const uint32\_t groupIdx)

*Starts the execution of a selected SW triggered ADC conversion group.*

- status\_t [ADC\\_StopGroupConversion](#) (const [adc\\_instance\\_t](#) \*const instance, const uint32\_t groupIdx, const uint32\_t timeout)

*Stops the selected SW triggered ADC conversion group execution.*

- status\_t [ADC\\_EnableNotification](#) (const [adc\\_instance\\_t](#) \*const instance, const uint32\_t groupIdx)

*Enables the notification callback for a configured group.*

- status\_t [ADC\\_DisableNotification](#) (const [adc\\_instance\\_t](#) \*const instance, const uint32\_t groupIdx)

*Disables the notification callback for a configured group.*

## 16.19.2 Data Structure Documentation

### 16.19.2.1 struct [adc\\_group\\_config\\_t](#)

Defines the configuration structure for an ADC PAL conversion group.

Implements : [adc\\_group\\_config\\_t](#)\_Class

Definition at line 129 of file [adc\\_pal.h](#).

#### Data Fields

- const [adc\\_input\\_chan\\_t](#) \* [inputChannelArray](#)
- uint16\_t \* [resultBuffer](#)
- uint8\_t [numChannels](#)
- uint8\_t [numSetsResultBuffer](#)
- bool [hwTriggerSupport](#)
- [adc\\_trigger\\_source\\_t](#) [triggerSource](#)
- [adc\\_delay\\_type\\_t](#) [delayType](#)
- uint16\_t \* [delayArray](#)
- bool [continuousConvEn](#)
- [adc\\_callback\\_t](#) [callback](#)
- void \* [callbackUserData](#)

#### Field Documentation

##### 16.19.2.1.1 [adc\\_callback\\_t](#) [callback](#)

Callback function associated with group conversion complete

Definition at line 145 of file [adc\\_pal.h](#).

##### 16.19.2.1.2 void\* [callbackUserData](#)

Pointer to additional user data to be passed by the callback

Definition at line 146 of file [adc\\_pal.h](#).

##### 16.19.2.1.3 bool [continuousConvEn](#)

Flag for enabling continuous conversions of a group - used only for SW triggered groups i.e. [hwTriggerSupport](#)==false.

Definition at line 143 of file [adc\\_pal.h](#).

**16.19.2.1.4 uint16\_t\* delayArray**

Pointer to array of delay values introduced from the occurrence of a HW trigger event until each ADC conversion in the group can start execution. Expressed in clock ticks. Note: the delay might be bigger if there is an overlap with another conversion already executing.

Definition at line 141 of file `adc_pal.h`.

**16.19.2.1.5 adc\_delay\_type\_t delayType**

Type of delay configuration. Supported values are platform dependent.

Definition at line 140 of file `adc_pal.h`.

**16.19.2.1.6 bool hwTriggerSupport**

Conversion group is HW triggered (true) or SW triggered (false).

Definition at line 137 of file `adc_pal.h`.

**16.19.2.1.7 const adc\_input\_chan\_t\* inputChannelArray**

Pointer to the array of ADC input channels. Each entry in this array corresponds to an individual conversion in the group. Only on some of the platforms the same input channel may appear multiple times - see device family specific details in the ADC PAL documentation.

Definition at line 131 of file `adc_pal.h`.

**16.19.2.1.8 uint8\_t numChannels**

Number of input channels in the array

Definition at line 134 of file `adc_pal.h`.

**16.19.2.1.9 uint8\_t numSetsResultBuffer**

Number of sets of results which can be stored in result buffer: length of the result buffer = numChannels x numSetsResultBuffer

Definition at line 135 of file `adc_pal.h`.

**16.19.2.1.10 uint16\_t\* resultBuffer**

Pointer to the array for conversion results

Definition at line 133 of file `adc_pal.h`.

**16.19.2.1.11 adc\_trigger\_source\_t triggerSource**

HW trigger source associated with the conversion group. Will be ignored if (`hwTriggerSupport == false`). Note for ADC\_SAR\_CTU: this enables the HW trigger source for all other groups; the actual order of execution of groups depends on the order of occurrence of triggers.

Definition at line 138 of file `adc_pal.h`.

**16.19.2.2 struct adc\_config\_t**

Defines the configuration structure for ADC PAL.

Implements : `adc_config_t_Class`

Definition at line 155 of file `adc_pal.h`.

**Data Fields**

- const `adc_group_config_t * groupConfigArray`

- uint16\_t [numGroups](#)
- uint8\_t [sampleTicks](#)
- void \* [extension](#)

#### Field Documentation

##### 16.19.2.2.1 void\* extension

This field is used to add extra IP specific settings to the basic configuration.

Definition at line 161 of file `adc_pal.h`.

##### 16.19.2.2.2 const adc\_group\_config\_t\* groupConfigArray

Array of group configurations

Definition at line 157 of file `adc_pal.h`.

##### 16.19.2.2.3 uint16\_t numGroups

Number of elements in groupConfigArray

Definition at line 158 of file `adc_pal.h`.

##### 16.19.2.2.4 uint8\_t sampleTicks

Duration of sample time expressed in ADC clock ticks

Definition at line 160 of file `adc_pal.h`.

##### 16.19.2.3 struct extension\_adc\_s32k1xx\_t

Defines the extension structure for ADC S32K1xx.

Implements : `extension_adc_s32k1xx_t_Class`

Definition at line 171 of file `adc_pal.h`.

#### Data Fields

- [adc\\_clk\\_divide\\_t](#) clockDivide
- [adc\\_resolution\\_t](#) resolution
- [adc\\_input\\_clock\\_t](#) inputClock
- [adc\\_voltage\\_reference\\_t](#) voltageRef
- bool [supplyMonitoringEnable](#)
- [pdb\\_clk\\_prescaler\\_div\\_t](#) pdbPrescaler

#### Field Documentation

##### 16.19.2.3.1 adc\_clk\_divide\_t clockDivide

Divider of the input clock for the ADC

Definition at line 173 of file `adc_pal.h`.

##### 16.19.2.3.2 adc\_input\_clock\_t inputClock

Input clock source

Definition at line 175 of file `adc_pal.h`.

##### 16.19.2.3.3 pdb\_clk\_prescaler\_div\_t pdbPrescaler

PDB clock prescaler. Delays are measured based on PDB clock divided by prescaler. Only relevant if delays are used.

Definition at line 178 of file adc\_pal.h.

#### 16.19.2.3.4 **adc\_resolution\_t** resolution

ADC resolution (8,10,12 bit)

Definition at line 174 of file adc\_pal.h.

#### 16.19.2.3.5 **bool** supplyMonitoringEnable

Enable internal supply monitoring

Definition at line 177 of file adc\_pal.h.

#### 16.19.2.3.6 **adc\_voltage\_reference\_t** voltageRef

Voltage reference used

Definition at line 176 of file adc\_pal.h.

### 16.19.3 Typedef Documentation

#### 16.19.3.1 **typedef** adc\_inputchannel\_t adc\_input\_chan\_t

Defines the enumeration with ADC PAL input channels.

Implements : adc\_input\_chan\_t\_Class

Definition at line 54 of file adc\_pal.h.

#### 16.19.3.2 **typedef** trgmux\_trigger\_source\_t adc\_trigger\_source\_t

Defines the enumeration with ADC PAL hardware trigger sources.

Implements : adc\_trigger\_source\_t\_Class

Definition at line 61 of file adc\_pal.h.

### 16.19.4 Enumeration Type Documentation

#### 16.19.4.1 **enum** adc\_delay\_type\_t

Defines an enumeration which contains the types of delay configurations for ADC conversions within a group.

Implements : adc\_delay\_type\_t\_Class

#### Enumerator

**ADC\_DELAY\_TYPE\_NO\_DELAY** First conversion can start right after the trigger occurrence, and the rest of conversions execute one after another

**ADC\_DELAY\_TYPE\_GROUP\_DELAY** Delay only first conversion, and the rest execute one after another

**ADC\_DELAY\_TYPE\_INDIVIDUAL\_DELAY** Individual delay for each conversion in the group (each measured from the occurrence of the trigger)

Definition at line 117 of file adc\_pal.h.

### 16.19.5 Function Documentation

#### 16.19.5.1 **status\_t** ADC\_Deinit ( **const** adc\_instance\_t \*const *instance* )

Deinitializes the ADC PAL instance.

This function resets the ADC PAL instance, including the other platform specific HW units used together with ADC, if there are no active conversions.

**Parameters**

<i>in</i>	<i>instance</i>	Pointer to ADC PAL instance number structure
-----------	-----------------	--

**Returns**

status:

- STATUS\_BUSY: there is already a HW triggered group enabled or executing, or a SW triggered group executing
- STATUS\_BUSY: on MPC574x platforms, if the BCTU module could not be reset
- STATUS\_SUCCESS: ADC PAL initialized successfully

Definition at line 338 of file `adc_pal.c`.

**16.19.5.2** `status_t ADC_DisableHardwareTrigger ( const adc_instance_t *const instance, const uint32_t groupidx, const uint32_t timeout )`

Disables the selected HW trigger for a conversion group, if the conversion group is HW triggered.

This function disables the HW trigger for a configured conversion group and also may stop its execution (depending on platform), if called when a conversion group is executing. If stopping is supported, the execution shall be stopped according to device specific procedures. The function shall wait for the procedures to complete within the given timeout interval and return error code if they do not succeed. : the function prevents new conversions from the group from starting, then waits until the current active conversion finishes execution (if the function call occurred while an ADC conversion from the group is executing) or timeout occurs. : the execution of a HW triggered group of conversions cannot be stopped, so the function shall wait until it is complete or timeout occurs. : the function always returns STATUS\_SUCCESS (even if a conversion is still executing) and doesn't use 'timeout' parameter. If it is called during a control cycle, between MRS and actual group conversion start, there will be an additional execution of the group, without callback.

**Parameters**

<i>in</i>	<i>instance</i>	Pointer to ADC PAL instance number structure
<i>in</i>	<i>groupidx</i>	Index of the selected group configured via groupConfigArray in <a href="#">adc_config_t</a>
<i>in</i>	<i>timeout</i>	Timeout interval in milliseconds

**Returns**

status:

- STATUS\_TIMEOUT: the operation did not complete successfully within the provided timeout interval
- STATUS\_SUCCESS: the operation completed successfully within the provided timeout interval

Definition at line 529 of file `adc_pal.c`.

**16.19.5.3** `status_t ADC_DisableNotification ( const adc_instance_t *const instance, const uint32_t groupidx )`

Disables the notification callback for a configured group.

This function disables the notification callback for a selected group of ADC conversions.

**Parameters**

<i>in</i>	<i>instance</i>	Pointer to ADC PAL instance number structure
<i>in</i>	<i>groupidx</i>	Index of the selected group configured via groupConfigArray in <a href="#">adc_config_t</a>

**Returns**

status:

- STATUS\_ERROR: the selected group is not active (SW triggered running or HW triggered running or enabled)
- STATUS\_SUCCESS: the notification has been disabled successfully

Definition at line 870 of file `adc_pal.c`.

**16.19.5.4** `status_t ADC_EnableHardwareTrigger ( const adc_instance_t *const instance, const uint32_t groupidx )`

Enables the selected HW trigger for a conversion group, if the conversion group has support for HW trigger.

Enables the selected HW trigger for a conversion group, if the conversion group has support for HW trigger. The function will return an error code if there is a conversion group already active. If the function succeeds, the conversion group will be triggered for execution when the selected HW trigger occurs.

**Parameters**

in	<i>instance</i>	Pointer to ADC PAL instance number structure
in	<i>groupidx</i>	Index of the selected group configured via groupConfigArray in <a href="#">adc_config_t</a>

**Returns**

status:

- STATUS\_BUSY: there is already a HW triggered group enabled or executing, or a SW triggered group executing
- STATUS\_SUCCESS: HW trigger enabled successfully for the selected conversion group

Definition at line 437 of file `adc_pal.c`.

**16.19.5.5** `status_t ADC_EnableNotification ( const adc_instance_t *const instance, const uint32_t groupidx )`

Enables the notification callback for a configured group.

This function enables the notification callback for a selected group of ADC conversions.

**Parameters**

in	<i>instance</i>	Pointer to ADC PAL instance number structure
in	<i>groupidx</i>	Index of the selected group configured via groupConfigArray in <a href="#">adc_config_t</a>

**Returns**

status:

- STATUS\_ERROR: the selected group is not active (SW triggered running or HW triggered running or enabled)
- STATUS\_SUCCESS: the notification has been enabled successfully

Definition at line 820 of file `adc_pal.c`.

**16.19.5.6** `status_t ADC_Init ( const adc_instance_t *const instance, const adc_config_t *const config )`

Initializes the ADC PAL instance.

This function initializes the ADC PAL instance, including the other platform specific HW units used together with ADC. Notifications are default enabled after init.

**Parameters**

in	<i>instance</i>	Pointer to ADC PAL instance number structure
in	<i>config</i>	The ADC PAL configuration structure

**Returns**

status:

- STATUS\_ERROR: platform specific error encountered while initializing one of the HW modules used by ADC PAL. On MPC574x returned if ADC calibration did not succeed for all the selected ADCs. On S32K1xx returned if it cannot reconfigure successfully the TRGMUX trigger source of the used PDB instance.
- STATUS\_BUSY: on MPC574x platforms, if the BCTU module could not be reset
- STATUS\_SUCCESS: ADC PAL initialized successfully

Definition at line 255 of file `adc_pal.c`.

#### 16.19.5.7 `status_t ADC_StartGroupConversion ( const adc_instance_t *const instance, const uint32_t groupidx )`

Starts the execution of a selected SW triggered ADC conversion group.

This function starts execution of a selected ADC conversion group, if there is no other conversion group active. Conversion groups started by `ADC_StartGroupConversion` shall not be preempted by HW triggered conversion groups.

##### Parameters

in	<i>instance</i>	Pointer to ADC PAL instance number structure
in	<i>groupidx</i>	Index of the selected group configured via groupConfigArray in <a href="#">adc_config_t</a>

##### Returns

status:

- STATUS\_BUSY: there is already a HW triggered group enabled or executing, or a SW triggered group executing
- STATUS\_SUCCESS: group conversion successfully triggered

Definition at line 640 of file `adc_pal.c`.

#### 16.19.5.8 `status_t ADC_StopGroupConversion ( const adc_instance_t *const instance, const uint32_t groupidx, const uint32_t timeout )`

Stops the selected SW triggered ADC conversion group execution.

This function stops the execution of a SW triggered conversion group. The execution shall be stopped according to device specific procedures. The function shall wait for the procedures to complete within the given timeout interval and return error code if they do not succeed. For `ADC_SAR_CTU` type and `MPC574xC_G_R` a conversion already started for execution cannot be stopped, so the function shall wait until it finishes or timeout occurs.

##### Parameters

in	<i>instance</i>	Pointer to ADC PAL instance number structure
in	<i>groupidx</i>	Index of the selected group configured via groupConfigArray in <a href="#">adc_config_t</a>
in	<i>timeout</i>	Timeout interval in milliseconds

##### Returns

status:

- STATUS\_TIMEOUT: the operation did not complete successfully within the provided timeout interval
- STATUS\_SUCCESS: the operation completed successfully within the provided timeout interval

Definition at line 720 of file `adc_pal.c`.



## 16.20 Automotive Math and Motor Control Library

Automotive Math and Motor Control Library integration with S32 SDK

### General Information

The Automotive Math and Motor Control Library Set is a precompiled off-the-shelf software library containing the building blocks for a wide range of motor control and general mathematical applications. The Automotive Math and Motor Control Library Set is delivered as a precompiled or source code library, and provides production-ready, highly speed-optimized, intensively tested and easy to use solution for the rapid development of user motor control and general mathematical applications. An integral part of the Automotive Math and Motor Control Library Set are the Matlab/Simulink® models of all supported functions to allow modelling of the user application using the Matlab/Simulink® environment, and extensive user documentation. The Automotive Math and Motor Control Library Set support three major arithmetic: 32-bit fixed-point, 16-bit fixed-point and single precision floating-point. For more information, application notes and demos please visit [AMMCLib page](#) from NXP website. AMMCLIB package contains the following:

- **bam** : contains Bit Accurate Models of all the functions for Matlab/Simulink
- **doc** : contains the User Manual
- **include** : contains all the header files, including the master header files of each library to be included in the user application
- **lib** : contains the compiled library file to be included in the user application

### Note

For an overview of what is included in the Automotive Math and Motor Control Library and its capabilities you can check the document found in [<SDK\\_Location>/doc/AMMCLIB\\_OnePager\\_S32SDK.pdf](#). This is just a brief description of the Automotive Math and Motor Control Library, for more information please check the full library documentation found in [<SDK\\_Location>/lib/<CPU\\_Family>/AMMCLIB/doc/S32K14XMCLUG.pdf](#). The library is provided in binary format, compiled using GCC, GHS, IAR and for evaluation purposes only. Please consult license.txt file for more information found in [<SDK\\_Location>/lib/<CPU\\_Family>/AMMCLIB/license.txt](#).

### How to use

To add AMMCLib in your application you need to follow four steps:

- 1) Add AMMCLib S32CT component into your project. The component will automatically add the required include paths.
- 2) Add ".S32K14x\_AMMCLIB.a" in Libraries (-l) and add "\${workspace\_loc:\${ProjName}/SDK/lib/AMMCLIB/lib/<compiler>}" in Library search path (-L)
- 3) Select the implementations from the AMMCLib component(Fixed 16, Fixed 32 or Float). If you are using the float implementation you need to enable FPU in the toolchain settings.
- 4) Use the library API to execute the required tests.

You can use the AMMCLib examples as a practical implementation of the steps described above.

### Note

IAR compiler:

- The compiler is not able to distinguish the sequence of read and write operations inside an inline assembly block in GFLIB\_VMin.h and it makes the Warning[Pe549] messages. In that case, C option "--diag\_suppress Pe549" should be added to suppress those warnings or "--warnings\_are\_errors" should be removed to not treat those warnings as errors.

## 16.21 Backward Compatibility Symbols for S32K142

This module covers backward compatibility symbols.

## 16.22 Bank N fail interrupt enable configuration options (0x38/0x48)

### 16.22.1 Detailed Description

This section lists all the possible options of 0x38/48 registers related to:

- HVIO-N short-circuit interrupt
- HVIO-N open load interrupt

#### Note

As HVIO bank 0 & 1 has the same memory layout, these macros can be used for Bank 0 & Bank 1 fail interrupt configuration

#### See also

[UJA113X\\_SetBankFailInterrupts\(\)](#)

#### HVIO4/8 short circuit interrupt configuration options (0x38:IO4SCIE[7] / 0x48:IO8SCIE[7])

- #define [IO48SCIE\\_DISABLE](#) 0x00U  
*Disable HVIO 4/8 short circuit interrupt.*
- #define [IO48SCIE\\_ENABLE](#) 0x80U  
*Enable HVIO 4/8 short circuit interrupt.*

#### HVIO4/8 open load interrupt configuration options (0x38:IO4OLIE[6] / 0x48:IO8OLIE[6])

- #define [IO48OLIE\\_DISABLE](#) 0x00U  
*Disable HVIO 4/8 open load interrupt.*
- #define [IO48OLIE\\_ENABLE](#) 0x40U  
*Enable HVIO 4/8 open load interrupt.*

#### HVIO3/7 short circuit interrupt configuration options (0x38:IO3SCIE[5] / 0x48:IO7SCIE[5])

- #define [IO37SCIE\\_DISABLE](#) 0x00U  
*Disable HVIO 3/7 short circuit interrupt.*
- #define [IO37SCIE\\_ENABLE](#) 0x20U  
*Enable HVIO 3/7 short circuit interrupt.*

#### HVIO3/7 open load interrupt configuration options (0x38:IO3OLIE[4] / 0x48:IO7OLIE[4])

- #define [IO37OLIE\\_DISABLE](#) 0x00U  
*Disable HVIO 3/7 open load interrupt.*
- #define [IO37OLIE\\_ENABLE](#) 0x10U  
*Enable HVIO 3/7 open load interrupt.*

#### HVIO2/6 short circuit interrupt configuration options (0x38:IO2SCIE[3] / 0x48:IO6SCIE[3])

- #define [IO26SCIE\\_DISABLE](#) 0x00U  
*Disable HVIO 2/6 short circuit interrupt.*
- #define [IO26SCIE\\_ENABLE](#) 0x08U  
*Enable HVIO 2/6 short circuit interrupt.*

#### HVIO2/6 open load interrupt configuration options (0x38:IO2OLIE[2] / 0x48:IO6OLIE[2])

- #define [IO26OLIE\\_DISABLE](#) 0x00U  
*Disable HVIO 2/6 open load interrupt.*
- #define [IO26OLIE\\_ENABLE](#) 0x04U  
*Enable HVIO 2/6 open load interrupt.*

#### HVIO1/5 short circuit interrupt configuration options (0x38:IO1SCIE[1] / 0x48:IO5SCIE[1])

- #define [IO15SCIE\\_DISABLE](#) 0x00U  
*Disable HVIO 1/5 short circuit interrupt.*
- #define [IO15SCIE\\_ENABLE](#) 0x02U  
*Enable HVIO 1/5 short circuit interrupt.*

#### HVIO1/5 open load interrupt configuration options (0x38:IO1OLIE[0] / 0x48:IO5OLIE[1])

- #define [IO15OLIE\\_DISABLE](#) 0x00U  
*Disable HVIO 1/5 open load interrupt.*
- #define [IO15OLIE\\_ENABLE](#) 0x01U  
*Enable HVIO 1/5 open load interrupt.*

### 16.22.2 Macro Definition Documentation

#### 16.22.2.1 #define IO15OLIE\_DISABLE 0x00U

Disable HVIO 1/5 open load interrupt.

Definition at line 1037 of file sbc\_uja113x.h.

#### 16.22.2.2 #define IO15OLIE\_ENABLE 0x01U

Enable HVIO 1/5 open load interrupt.

Definition at line 1038 of file sbc\_uja113x.h.

#### 16.22.2.3 #define IO15SCIE\_DISABLE 0x00U

Disable HVIO 1/5 short circuit interrupt.

Definition at line 1030 of file sbc\_uja113x.h.

#### 16.22.2.4 #define IO15SCIE\_ENABLE 0x02U

Enable HVIO 1/5 short circuit interrupt.

Definition at line 1031 of file sbc\_uja113x.h.

#### 16.22.2.5 #define IO26OLIE\_DISABLE 0x00U

Disable HVIO 2/6 open load interrupt.

Definition at line 1023 of file sbc\_uja113x.h.

#### 16.22.2.6 #define IO26OLIE\_ENABLE 0x04U

Enable HVIO 2/6 open load interrupt.

Definition at line 1024 of file sbc\_uja113x.h.

**16.22.2.7 #define IO26SCIE\_DISABLE 0x00U**

Disable HVIO 2/6 short circuit interrupt.

Definition at line 1016 of file sbc\_uja113x.h.

**16.22.2.8 #define IO26SCIE\_ENABLE 0x08U**

Enable HVIO 2/6 short circuit interrupt.

Definition at line 1017 of file sbc\_uja113x.h.

**16.22.2.9 #define IO37OLIE\_DISABLE 0x00U**

Disable HVIO 3/7 open load interrupt.

Definition at line 1009 of file sbc\_uja113x.h.

**16.22.2.10 #define IO37OLIE\_ENABLE 0x10U**

Enable HVIO 3/7 open load interrupt.

Definition at line 1010 of file sbc\_uja113x.h.

**16.22.2.11 #define IO37SCIE\_DISABLE 0x00U**

Disable HVIO 3/7 short circuit interrupt.

Definition at line 1002 of file sbc\_uja113x.h.

**16.22.2.12 #define IO37SCIE\_ENABLE 0x20U**

Enable HVIO 3/7 short circuit interrupt.

Definition at line 1003 of file sbc\_uja113x.h.

**16.22.2.13 #define IO48OLIE\_DISABLE 0x00U**

Disable HVIO 4/8 open load interrupt.

Definition at line 995 of file sbc\_uja113x.h.

**16.22.2.14 #define IO48OLIE\_ENABLE 0x40U**

Enable HVIO 4/8 open load interrupt.

Definition at line 996 of file sbc\_uja113x.h.

**16.22.2.15 #define IO48SCIE\_DISABLE 0x00U**

Disable HVIO 4/8 short circuit interrupt.

Definition at line 988 of file sbc\_uja113x.h.

**16.22.2.16 #define IO48SCIE\_ENABLE 0x80U**

Enable HVIO 4/8 short circuit interrupt.

Definition at line 989 of file sbc\_uja113x.h.

## 16.23 Bank N wake-up interrupt enable configuration options (0x37/0x47)

### 16.23.1 Detailed Description

This section lists all the possible options of 0x37/47 registers related to:

- HVIO-N falling edge interrupt
- HVIO-N rising edge interrupt

#### Note

As HVIO bank 0 & 1 has the same memory layout, these macros can be used for Bank 0 & Bank 1 wake-up interrupt configuration

#### See also

[UJA113X\\_SetBankWupInterrupts\(\)](#)

#### HVIO4/8 falling edge interrupt configuration options (0x37:IO4FEIE[7]/0x47:IO8FEIE[7])

- #define [IO48FEIE\\_DISABLE](#) 0x00U  
*Disable HVIO 4/8 falling edge interrupt.*
- #define [IO48FEIE\\_ENABLE](#) 0x80U  
*Enable HVIO 4/8 falling edge interrupt.*

#### HVIO4/8 rising edge interrupt configuration options (0x37:IO4REIE[6]/0x47:IO8REIE[6])

- #define [IO48REIE\\_DISABLE](#) 0x00U  
*Disable HVIO 4/8 rising edge interrupt.*
- #define [IO48REIE\\_ENABLE](#) 0x40U  
*Enable HVIO 4/8 rising edge interrupt.*

#### HVIO3/7 falling edge interrupt configuration options (0x37:IO3FEIE[5]/0x47:IO7FEIE[5])

- #define [IO37FEIE\\_DISABLE](#) 0x00U  
*Disable HVIO 3/7 falling edge interrupt.*
- #define [IO37FEIE\\_ENABLE](#) 0x20U  
*Enable HVIO 3/7 falling edge interrupt.*

#### HVIO3/7 rising edge interrupt configuration options (0x37:IO3REIE[4]/0x47:IO7REIE[4])

- #define [IO37REIE\\_DISABLE](#) 0x00U  
*Disable HVIO 3/7 rising edge interrupt.*
- #define [IO37REIE\\_ENABLE](#) 0x10U  
*Enable HVIO 3/7 rising edge interrupt.*

#### HVIO2/6 falling edge interrupt configuration options (0x37:IO2FEIE[3]/0x47:IO6FEIE[3])

- #define [IO26FEIE\\_DISABLE](#) 0x00U  
*Disable HVIO 2/6 falling edge interrupt.*
- #define [IO26FEIE\\_ENABLE](#) 0x08U  
*Enable HVIO 2/6 falling edge interrupt.*

## HVIO2/6 rising edge interrupt configuration options (0x37:IO2REIE[2]/0x47:IO6[REIE[2]])

- #define `IO26REIE_DISABLE` 0x00U  
*Disable HVIO 2/6 rising edge interrupt.*
- #define `IO26REIE_ENABLE` 0x04U  
*Enable HVIO 2/6 rising edge interrupt.*

## HVIO1/5 falling edge interrupt configuration options (0x37:IO1FEIE[1]/0x47:IO5FEIE[1])

- #define `IO15FEIE_DISABLE` 0x00U  
*Disable HVIO 1/5 falling edge interrupt.*
- #define `IO15FEIE_ENABLE` 0x02U  
*Enable HVIO 1/5 falling edge interrupt.*

## HVIO1/5 rising edge interrupt configuration options (0x37:IO1REIE[0]/0x47:IO1REIE[0])

- #define `IO15REIE_DISABLE` 0x00U  
*Disable HVIO 1/5 rising edge interrupt.*
- #define `IO15REIE_ENABLE` 0x01U  
*Enable HVIO 1/5 rising edge interrupt.*

## 16.23.2 Macro Definition Documentation

16.23.2.1 #define `IO15FEIE_DISABLE` 0x00U

Disable HVIO 1/5 falling edge interrupt.

Definition at line 957 of file `sbc_uja113x.h`.

16.23.2.2 #define `IO15FEIE_ENABLE` 0x02U

Enable HVIO 1/5 falling edge interrupt.

Definition at line 958 of file `sbc_uja113x.h`.

16.23.2.3 #define `IO15REIE_DISABLE` 0x00U

Disable HVIO 1/5 rising edge interrupt.

Definition at line 964 of file `sbc_uja113x.h`.

16.23.2.4 #define `IO15REIE_ENABLE` 0x01U

Enable HVIO 1/5 rising edge interrupt.

Definition at line 965 of file `sbc_uja113x.h`.

16.23.2.5 #define `IO26FEIE_DISABLE` 0x00U

Disable HVIO 2/6 falling edge interrupt.

Definition at line 943 of file `sbc_uja113x.h`.

16.23.2.6 #define `IO26FEIE_ENABLE` 0x08U

Enable HVIO 2/6 falling edge interrupt.

Definition at line 944 of file `sbc_uja113x.h`.

**16.23.2.7 #define IO26REIE\_DISABLE 0x00U**

Disable HVIO 2/6 rising edge interrupt.

Definition at line 950 of file sbc\_uja113x.h.

**16.23.2.8 #define IO26REIE\_ENABLE 0x04U**

Enable HVIO 2/6 rising edge interrupt.

Definition at line 951 of file sbc\_uja113x.h.

**16.23.2.9 #define IO37FEIE\_DISABLE 0x00U**

Disable HVIO 3/7 falling edge interrupt.

Definition at line 929 of file sbc\_uja113x.h.

**16.23.2.10 #define IO37FEIE\_ENABLE 0x20U**

Enable HVIO 3/7 falling edge interrupt.

Definition at line 930 of file sbc\_uja113x.h.

**16.23.2.11 #define IO37REIE\_DISABLE 0x00U**

Disable HVIO 3/7 rising edge interrupt.

Definition at line 936 of file sbc\_uja113x.h.

**16.23.2.12 #define IO37REIE\_ENABLE 0x10U**

Enable HVIO 3/7 rising edge interrupt.

Definition at line 937 of file sbc\_uja113x.h.

**16.23.2.13 #define IO48FEIE\_DISABLE 0x00U**

Disable HVIO 4/8 falling edge interrupt.

Definition at line 915 of file sbc\_uja113x.h.

**16.23.2.14 #define IO48FEIE\_ENABLE 0x80U**

Enable HVIO 4/8 falling edge interrupt.

Definition at line 916 of file sbc\_uja113x.h.

**16.23.2.15 #define IO48REIE\_DISABLE 0x00U**

Disable HVIO 4/8 rising edge interrupt.

Definition at line 922 of file sbc\_uja113x.h.

**16.23.2.16 #define IO48REIE\_ENABLE 0x40U**

Enable HVIO 4/8 rising edge interrupt.

Definition at line 923 of file sbc\_uja113x.h.



## 16.24 CRC Driver

### 16.24.1 Detailed Description

Cyclic Redundancy Check Peripheral Driver.

This section describes the programming interface of the CRC driver.

#### Data Structures

- struct [crc\\_user\\_config\\_t](#)  
CRC configuration structure. Implements : [crc\\_user\\_config\\_t\\_Class](#). [More...](#)

#### Enumerations

- enum [crc\\_transpose\\_t](#) { [CRC\\_TRANSPOSE\\_NONE](#) = 0x00U, [CRC\\_TRANSPOSE\\_BITS](#) = 0x01U, [CRC\\_TRANSPOSE\\_BITS\\_AND\\_BYTES](#) = 0x02U, [CRC\\_TRANSPOSE\\_BYTES](#) = 0x03U }  
CRC type of transpose of read write data Implements : [crc\\_transpose\\_t\\_Class](#).

#### CRC DRIVER API

- status\_t [CRC\\_DRV\\_Init](#) (uint32\_t instance, const [crc\\_user\\_config\\_t](#) \*userConfigPtr)  
*Initializes the CRC module.*
- status\_t [CRC\\_DRV\\_Deinit](#) (uint32\_t instance)  
*Sets the default configuration.*
- uint32\_t [CRC\\_DRV\\_GetCrc32](#) (uint32\_t instance, uint32\_t data, bool newSeed, uint32\_t seed)  
*Appends 32-bit data to the current CRC calculation and returns new result.*
- uint32\_t [CRC\\_DRV\\_GetCrc16](#) (uint32\_t instance, uint16\_t data, bool newSeed, uint32\_t seed)  
*Appends 16-bit data to the current CRC calculation and returns new result.*
- uint32\_t [CRC\\_DRV\\_GetCrc8](#) (uint32\_t instance, uint8\_t data, bool newSeed, uint32\_t seed)  
*Appends 8-bit data to the current CRC calculation and returns new result.*
- void [CRC\\_DRV\\_WriteData](#) (uint32\_t instance, const uint8\_t \*data, uint32\_t dataSize)  
*Appends a block of bytes to the current CRC calculation.*
- uint32\_t [CRC\\_DRV\\_GetCrcResult](#) (uint32\_t instance)  
*Returns the current result of the CRC calculation.*
- status\_t [CRC\\_DRV\\_Configure](#) (uint32\_t instance, const [crc\\_user\\_config\\_t](#) \*userConfigPtr)  
*Configures the CRC module from a user configuration structure.*
- status\_t [CRC\\_DRV\\_GetConfig](#) (uint32\_t instance, [crc\\_user\\_config\\_t](#) \*const userConfigPtr)  
*Get configures of the CRC module currently.*
- status\_t [CRC\\_DRV\\_GetDefaultConfig](#) ([crc\\_user\\_config\\_t](#) \*const userConfigPtr)  
*Get default configures the CRC module for configuration structure.*

### 16.24.2 Data Structure Documentation

#### 16.24.2.1 struct [crc\\_user\\_config\\_t](#)

CRC configuration structure. Implements : [crc\\_user\\_config\\_t\\_Class](#).

Definition at line 83 of file [crc\\_driver.h](#).

#### Data Fields

- [crc\\_transpose\\_t](#) [writeTranspose](#)
- bool [complementChecksum](#)
- uint32\_t [seed](#)

## Field Documentation

### 16.24.2.1.1 `bool complementChecksum`

True if the result shall be complement of the actual checksum.

Definition at line 95 of file `crc_driver.h`.

### 16.24.2.1.2 `uint32_t seed`

Starting checksum value.

Definition at line 96 of file `crc_driver.h`.

### 16.24.2.1.3 `crc_transpose_t writeTranspose`

Type of transpose when writing CRC input data.

Definition at line 94 of file `crc_driver.h`.

## 16.24.3 Enumeration Type Documentation

### 16.24.3.1 `enum crc_transpose_t`

CRC type of transpose of read write data Implements : `crc_transpose_t_Class`.

## Enumerator

**`CRC_TRANSPOSE_NONE`** No transpose

**`CRC_TRANSPOSE_BITS`** Transpose bits in bytes

**`CRC_TRANSPOSE_BITS_AND_BYTES`** Transpose bytes and bits in bytes

**`CRC_TRANSPOSE_BYTES`** Transpose bytes

Definition at line 44 of file `crc_driver.h`.

## 16.24.4 Function Documentation

### 16.24.4.1 `status_t CRC_DRV_Configure ( uint32_t instance, const crc_user_config_t * userConfigPtr )`

Configures the CRC module from a user configuration structure.

This function configures the CRC module from a user configuration structure

## Parameters

<code>in</code>	<code>instance</code>	The CRC instance number
<code>in</code>	<code>userConfigPtr</code>	Pointer to structure of initialization

## Returns

Execution status (success)

Definition at line 236 of file `crc_driver.c`.

### 16.24.4.2 `status_t CRC_DRV_Deinit ( uint32_t instance )`

Sets the default configuration.

This function sets the default configuration

**Parameters**

in	<i>instance</i>	The CRC instance number
----	-----------------	-------------------------

**Returns**

Execution status (success)

Definition at line 88 of file `crc_driver.c`.

#### 16.24.4.3 `status_t CRC_DRV_GetConfig ( uint32_t instance, crc_user_config_t *const userConfigPtr )`

Get configures of the CRC module currently.

This function Get configures of the CRC module currently

**Parameters**

in	<i>instance</i>	The CRC instance number
out	<i>userConfigPtr</i>	Pointer to structure of initialization

**Returns**

Execution status (success)

Definition at line 268 of file `crc_driver.c`.

#### 16.24.4.4 `uint32_t CRC_DRV_GetCrc16 ( uint32_t instance, uint16_t data, bool newSeed, uint32_t seed )`

Appends 16-bit data to the current CRC calculation and returns new result.

This function appends 16-bit data to the current CRC calculation and returns new result. If the `newSeed` is true, seed set and result are calculated from the seed new value (new CRC calculation)

**Parameters**

in	<i>instance</i>	The CRC instance number
in	<i>data</i>	Input data for CRC calculation
in	<i>newSeed</i>	Sets new CRC calculation <ul style="list-style-type: none"> <li>• true: New seed set and used for new calculation.</li> <li>• false: Seed argument ignored, continues old calculation.</li> </ul>
in	<i>seed</i>	New seed if <code>newSeed</code> is true, else ignored

**Returns**

New CRC result

Definition at line 139 of file `crc_driver.c`.

#### 16.24.4.5 `uint32_t CRC_DRV_GetCrc32 ( uint32_t instance, uint32_t data, bool newSeed, uint32_t seed )`

Appends 32-bit data to the current CRC calculation and returns new result.

This function appends 32-bit data to the current CRC calculation and returns new result. If the `newSeed` is true, seed set and result are calculated from the seed new value (new CRC calculation)

**Parameters**

in	<i>instance</i>	The CRC instance number
in	<i>data</i>	Input data for CRC calculation
in	<i>newSeed</i>	Sets new CRC calculation <ul style="list-style-type: none"> <li>• true: New seed set and used for new calculation.</li> <li>• false: Seed argument ignored, continues old calculation.</li> </ul>
in	<i>seed</i>	New seed if newSeed is true, else ignored

**Returns**

New CRC result

Definition at line 108 of file `crc_driver.c`.

**16.24.4.6** `uint32_t CRC_DRV_GetCrc8 ( uint32_t instance, uint8_t data, bool newSeed, uint32_t seed )`

Appends 8-bit data to the current CRC calculation and returns new result.

This function appends 8-bit data to the current CRC calculation and returns new result. If the newSeed is true, seed set and result are calculated from the seed new value (new CRC calculation)

**Parameters**

in	<i>instance</i>	The CRC instance number
in	<i>data</i>	Input data for CRC calculation
in	<i>newSeed</i>	Sets new CRC calculation <ul style="list-style-type: none"> <li>• true: New seed set and used for new calculation.</li> <li>• false: Seed argument ignored, continues old calculation.</li> </ul>
in	<i>seed</i>	New seed if newSeed is true, else ignored

**Returns**

New CRC result

Definition at line 169 of file `crc_driver.c`.

**16.24.4.7** `uint32_t CRC_DRV_GetCrcResult ( uint32_t instance )`

Returns the current result of the CRC calculation.

This function returns the current result of the CRC calculation

**Parameters**

in	<i>instance</i>	The CRC instance number
----	-----------------	-------------------------

**Returns**

Result of CRC calculation

Definition at line 220 of file `crc_driver.c`.

**16.24.4.8** `status_t CRC_DRV_GetDefaultConfig ( crc_user_config_t *const userConfigPtr )`

Get default configures the CRC module for configuration structure.

This function Get default configures the CRC module for user configuration structure

**Parameters**

out	<i>userConfigPtr</i>	Pointer to structure of initialization
-----	----------------------	--

**Returns**

Execution status (success)

Definition at line 300 of file `crc_driver.c`.

**16.24.4.9** `status_t CRC_DRV_Init ( uint32_t instance, const crc_user_config_t * userConfigPtr )`

Initializes the CRC module.

This function initializes CRC driver based on user configuration input. The user must make sure that the clock is enabled

**Parameters**

in	<i>instance</i>	The CRC instance number
in	<i>userConfigPtr</i>	Pointer to structure of initialization

**Returns**

Execution status (success)

Definition at line 65 of file `crc_driver.c`.

**16.24.4.10** `void CRC_DRV_WriteData ( uint32_t instance, const uint8_t * data, uint32_t dataSize )`

Appends a block of bytes to the current CRC calculation.

This function appends a block of bytes to the current CRC calculation

**Parameters**

in	<i>instance</i>	The CRC instance number
in	<i>data</i>	Data for current CRC calculation
in	<i>dataSize</i>	Length of data to be calculated

Definition at line 197 of file `crc_driver.c`.

## 16.25 CSEc Driver

### 16.25.1 Detailed Description

Cryptographic Services Engine Peripheral Driver.

#### How to use the CSEc driver in your application

To access the command feature set, the part must be configured for EEE operation, using the PGMPART command. This can be implemented by using the Flash driver. By enabling security features and configuring a number of user keys, the total size of the 4 KByte EEERAM will be reduced by the space required to store the user keys. The user key space will then effectively be unaddressable space in the EEERAM.

At the bottom of this page is an example of making this configuration using the Flash driver. For more details related to the FLASH\_DRV\_DEFlashPartition function, please refer to the Flash driver documentation. Please note that this configuration is required only once and should not be launched from Flash memory.

In order to use the CSEc driver in your application, the **CSEC\_DRV\_Init** function should be called prior to using the rest of the API. The parameter of this function is used for holding the internal state of the driver throughout the lifetime of the application.

#### Key/seed/random number generation

This is the high level flow in which to initialize and generate random numbers.

1. Run **CSEC\_DRV\_InitTRNG** to initialize a random seed from the internal TRNG
  - **CSEC\_DRV\_InitTRNG** must be run after every POR, and before the first execution of **CSEC\_DRV\_GenerateRND**
  - Note that if the next step (run **CSEC\_DRV\_GenerateRND**) is run without initializing the seed, **CSEC\_DRV\_RNG\_SEED** will be returned.
2. Run **CSEC\_DRV\_GenerateRND** to generate a random number. The PRNG uses the PRNG\_STATE/KEY and Seed per SHE spec and the AIS20 standard.
3. For additional random numbers the user may continue executing **CSEC\_DRV\_GenerateRND** unless a POR event occurred.

#### Memory update protocol

In order to update a key, the user must have knowledge of a valid authentication secret, i.e. another key (AuthID). If the key AuthID is empty, the key update will only work if AuthID = ID (the key that will be updated will represent the AuthID from now on), otherwise **CSEC\_KEY\_EMPTY** is returned.

The M1-M3 values need to be computed according to the SHE Specification in order to update a key slot. The **CSEC\_DRV\_LoadKey** function will require those values. After successfully updating the key slot, two verification values will be returned: M4 and M5. The user can compute the two values and compare them with the ones returned by the **CSEC\_DRV\_LoadKey** function in order to ensure the slot was updated as desired. Please refer to the CSEc driver example for a reference implementation of the memory update protocol.

#### Integration guideline

##### Compilation units

The following files need to be compiled in the project:

```
{S32SDK_PATH}\platform\drivers\src\csec\csec_driver.c
{S32SDK_PATH}\platform\drivers\src\csec\csec_hw_access.c
```

##### Include path

The following paths need to be added to the include path of the toolchain:

```
{S32SDK_PATH}\platform\drivers\inc\
```

## Preprocessor symbols

No special symbols are required for this component

## Important Note

While executing CSEC\_DRV\_GenerateMACAddrMode and CSEC\_DRV\_VerifyMACAddrMode functions, it is not possible to execute code from the FLASH block targeted by the current operation. This includes interrupt handlers for any interrupt that might occur during this time. It is the responsibility of the application to ensure that any such code is placed in a different FLASH block or in RAM. Functions can be placed in RAM section by using the START/END\_FUNCTION\_DEFINITION/DECLARATION\_RAMSECTION macros.

## Dependencies

### Interrupt Manager (Interrupt) OS Interface (OSIF)

## Examples:

Using the Flash driver to partition Flash for CSEc operation, the below code section applies for S32K14x:

```
flash_ssd_config_t flashSSDConfig;

FLASH_DRV_Init(&flashl_InitConfig0, &flashSSDConfig);

/* Configure the part for EEE operation, with 20 keys for CSEc */
FLASH_DRV_DEFlashPartition(&flashSSDConfig, 0x2, 0x4, 0x3, false, true);
```

### The example partition code for S32K11x:

```
flash_ssd_config_t flashSSDConfig;

FLASH_DRV_Init(&flashl_InitConfig0, &flashSSDConfig);

/* Configure the part for EEE operation, with 20 keys for CSEc */
FLASH_DRV_DEFlashPartition(&flashSSDConfig, 0x3, 0x3, 0x3, false, true);
```

## Encryption using AES EBC mode

```
uint8_t plainText[16] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88,
    0x99, 0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF};
uint8_t plainKey[16] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
    0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};

csec_error_code_t stat;
uint8_t cipherText[16];

csec_state_t csecState;

CSEC_DRV_Init(&csecState);

stat = CSEC_DRV_LoadPlainKey(plainKey);
if (stat != CSEC_NO_ERROR)
{
    /* Loading the key failed, encryption will not have the expected result */
    return false;
}

stat = CSEC_DRV_EncryptECB(CSEC_RAM_KEY, plainText, 16U, cipherText, 1U);
if (stat != CSEC_NO_ERROR)
{
    /* Encryption was successful */
    return true;
}
```

## Generating and verifying CMAC for a message

```
uint8_t plainKey[16] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6, 0xab,
    0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
uint8_t msg[16] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08,
    0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};
uint8_t cmac[16];
bool verifStatus;
csec_error_code_t stat;
```

```

csec_state_t csecState;

CSEC_DRV_Init(&csecState);

stat = CSEC_DRV_LoadPlainKey(plainKey);
if (stat != CSEC_NO_ERROR)
    return false;

stat = CSEC_DRV_GenerateMAC(CSEC_RAM_KEY, msg, 128U, cmac, 1U);
if (stat != CSEC_NO_ERROR)
    return false;

stat = CSEC_DRV_VerifyMAC(CSEC_RAM_KEY, msg, 128U, cmac, 128U, &verifStatus,
    1U);
if (stat != CSEC_NO_ERROR)
    return false;

if (!verifStatus)
{
    /* The given CMAC did not matched with the one computed internally */
    return false;
}

```

## Generating random bits

```

csec_error_code_t stat;
csec_status_t status;
uint8_t rnd[16];

csec_state_t csecState;

CSEC_DRV_Init(&csecState);

stat = CSEC_DRV_InitRNG();
if (stat != CSEC_NO_ERROR)
    return false;

/* Check RNG is initialized */
status = CSEC_DRV_GetStatus();
if (!(status & CSEC_STATUS_RND_INIT))
    return false;

stat = CSEC_DRV_GenerateRND(rnd);
if (stat != CSEC_NO_ERROR)
    return false;

```

## Data Structures

- struct `csec_state_t`  
Internal driver state information. [More...](#)

## Macros

- #define `CSEC_STATUS_BUSY` (0x1U)  
The bit is set whenever SHE is processing a command.
- #define `CSEC_STATUS_SECURE_BOOT` (0x2U)  
The bit is set if the secure booting is activated.
- #define `CSEC_STATUS_BOOT_INIT` (0x4U)  
The bit is set if the secure booting has been personalized during the boot sequence.
- #define `CSEC_STATUS_BOOT_FINISHED` (0x8U)  
The bit is set when the secure booting has been finished by calling either `CMD_BOOT_FAILURE` or `CMD_BOOT_OK` or if `CMD_SECURE_BOOT` failed in verifying `BOOT_MAC`.
- #define `CSEC_STATUS_BOOT_OK` (0x10U)  
The bit is set if the secure booting (`CMD_SECURE_BOOT`) succeeded. If `CMD_BOOT_FAILURE` is called the bit is erased.
- #define `CSEC_STATUS_RND_INIT` (0x20U)  
The bit is set if the random number generator has been initialized.
- #define `CSEC_STATUS_EXT_DEBUGGER` (0x40U)



The bit is set if an external debugger is connected to the chip.

- #define `CSEC_STATUS_INT_DEBUGGER` (0x80U)

The bit is set if the internal debugging mechanisms of SHE are activated.

## Typedefs

- typedef uint8\_t `csec_status_t`

Represents the status of the CSEc module. Provides one bit for each status code as per SHE specification. `CSEC_STATUS_*` masks can be used for verifying the status.

## Enumerations

- enum `csec_key_id_t` {  
`CSEC_SECRET_KEY` = 0x0U, `CSEC_MASTER_ECU`, `CSEC_BOOT_MAC_KEY`, `CSEC_BOOT_MAC`,  
`CSEC_KEY_1`, `CSEC_KEY_2`, `CSEC_KEY_3`, `CSEC_KEY_4`,  
`CSEC_KEY_5`, `CSEC_KEY_6`, `CSEC_KEY_7`, `CSEC_KEY_8`,  
`CSEC_KEY_9`, `CSEC_KEY_10`, `CSEC_RAM_KEY` = 0xFU, `CSEC_KEY_11` = 0x14U,  
`CSEC_KEY_12`, `CSEC_KEY_13`, `CSEC_KEY_14`, `CSEC_KEY_15`,  
`CSEC_KEY_16`, `CSEC_KEY_17` }  
Specify the KeyID to be used to implement the requested cryptographic operation.
- enum `csec_cmd_t` {  
`CSEC_CMD_ENC_ECB` = 0x1U, `CSEC_CMD_ENC_CBC`, `CSEC_CMD_DEC_ECB`, `CSEC_CMD_DEC_CBC`,  
`CSEC_CMD_GENERATE_MAC`, `CSEC_CMD_VERIFY_MAC`, `CSEC_CMD_LOAD_KEY`, `CSEC_CMD_LOAD_PLAIN_KEY`,  
`CSEC_CMD_EXPORT_RAM_KEY`, `CSEC_CMD_INIT_RNG`, `CSEC_CMD_EXTEND_SEED`, `CSEC_CMD_D_RND`,  
`CSEC_CMD_RESERVED_1`, `CSEC_CMD_BOOT_FAILURE`, `CSEC_CMD_BOOT_OK`, `CSEC_CMD_GET_ID`,  
`CSEC_CMD_BOOT_DEFINE`, `CSEC_CMD_DBG_CHAL`, `CSEC_CMD_DBG_AUTH`, `CSEC_CMD_RESERVED_2`,  
`CSEC_CMD_RESERVED_3`, `CSEC_CMD_MP_COMPRESS` }  
CSEc commands which follow the same values as the SHE command definition.
- enum `csec_call_sequence_t` { `CSEC_CALL_SEQ_FIRST`, `CSEC_CALL_SEQ_SUBSEQUENT` }  
Specifies if the information is the first or a following function call.
- enum `csec_boot_flavor_t` { `CSEC_BOOT_STRICT`, `CSEC_BOOT_SERIAL`, `CSEC_BOOT_PARALLEL`, `CSEC_BOOT_NOT_DEFINED` }  
Specifies the boot type for the `BOOT_DEFINE` command.

## Functions

- void `CSEC_DRV_Init` (`csec_state_t` \*state)  
Initializes the internal state of the driver and enables the FTFC interrupt.
- void `CSEC_DRV_Deinit` (void)  
Clears the internal state of the driver and disables the FTFC interrupt.
- status\_t `CSEC_DRV_EncryptECB` (`csec_key_id_t` keyId, const uint8\_t \*plainText, uint32\_t length, uint8\_t \*cipherText, uint32\_t timeout)  
Performs the AES-128 encryption in ECB mode.
- status\_t `CSEC_DRV_DecryptECB` (`csec_key_id_t` keyId, const uint8\_t \*cipherText, uint32\_t length, uint8\_t \*plainText, uint32\_t timeout)  
Performs the AES-128 decryption in ECB mode.
- status\_t `CSEC_DRV_EncryptCBC` (`csec_key_id_t` keyId, const uint8\_t \*plainText, uint32\_t length, const uint8\_t \*iv, uint8\_t \*cipherText, uint32\_t timeout)

- Performs the AES-128 encryption in CBC mode.*

  - status\_t [CSEC\\_DRV\\_DecryptCBC](#) (csec\_key\_id\_t keyId, const uint8\_t \*cipherText, uint32\_t length, const uint8\_t \*iv, uint8\_t \*plainText, uint32\_t timeout)
- Performs the AES-128 decryption in CBC mode.*

  - status\_t [CSEC\\_DRV\\_GenerateMAC](#) (csec\_key\_id\_t keyId, const uint8\_t \*msg, uint32\_t msgLen, uint8\_t \*mac, uint32\_t timeout)
- Calculates the MAC of a given message using CMAC with AES-128.*

  - status\_t [CSEC\\_DRV\\_GenerateMACAddrMode](#) (csec\_key\_id\_t keyId, const uint8\_t \*msg, uint32\_t msgLen, uint8\_t \*mac)
- Calculates the MAC of a given message (located in Flash) using CMAC with AES-128.*

  - status\_t [CSEC\\_DRV\\_VerifyMAC](#) (csec\_key\_id\_t keyId, const uint8\_t \*msg, uint32\_t msgLen, const uint8\_t \*mac, uint16\_t macLen, bool \*verifStatus, uint32\_t timeout)
- Verifies the MAC of a given message using CMAC with AES-128.*

  - status\_t [CSEC\\_DRV\\_VerifyMACAddrMode](#) (csec\_key\_id\_t keyId, const uint8\_t \*msg, uint32\_t msgLen, const uint8\_t \*mac, uint16\_t macLen, bool \*verifStatus)
- Verifies the MAC of a given message (located in Flash) using CMAC with AES-128.*

  - status\_t [CSEC\\_DRV\\_LoadKey](#) (csec\_key\_id\_t keyId, const uint8\_t \*m1, const uint8\_t \*m2, const uint8\_t \*m3, uint8\_t \*m4, uint8\_t \*m5)
- Updates an internal key per the SHE specification.*

  - status\_t [CSEC\\_DRV\\_LoadPlainKey](#) (const uint8\_t \*plainKey)
- Updates the RAM key memory slot with a 128-bit plaintext.*

  - status\_t [CSEC\\_DRV\\_ExportRAMKey](#) (uint8\_t \*m1, uint8\_t \*m2, uint8\_t \*m3, uint8\_t \*m4, uint8\_t \*m5)
- Exports the RAM\_KEY into a format protected by SECRET\_KEY.*

  - status\_t [CSEC\\_DRV\\_InitRNG](#) (void)
- Initializes the seed and derives a key for the PRNG.*

  - status\_t [CSEC\\_DRV\\_ExtendSeed](#) (const uint8\_t \*entropy)
- Extends the seed of the PRNG.*

  - status\_t [CSEC\\_DRV\\_GenerateRND](#) (uint8\_t \*rnd)
- Generates a vector of 128 random bits.*

  - status\_t [CSEC\\_DRV\\_BootFailure](#) (void)
- Signals a failure detected during later stages of the boot process.*

  - status\_t [CSEC\\_DRV\\_BootOK](#) (void)
- Marks a successful boot verification during later stages of the boot process.*

  - status\_t [CSEC\\_DRV\\_BootDefine](#) (uint32\_t bootSize, csec\_boot\_flavor\_t bootFlavor)
- Implements an extension of the SHE standard to define both the user boot size and boot method.*

  - static csec\_status\_t [CSEC\\_DRV\\_GetStatus](#) (void)
- Returns the content of the status register.*

  - status\_t [CSEC\\_DRV\\_GetID](#) (const uint8\_t \*challenge, uint8\_t \*uid, uint8\_t \*sreg, uint8\_t \*mac)
- Returns the identity (UID) and the value of the status register protected by a MAC over a challenge and the data.*

  - status\_t [CSEC\\_DRV\\_DbgChal](#) (uint8\_t \*challenge)
- Obtains a random number which the user shall use along with the MASTER\_ECU\_KEY and UID to return an authorization request.*

  - status\_t [CSEC\\_DRV\\_DbgAuth](#) (const uint8\_t \*authorization)
- Erases all keys (actual and outdated) stored in NVM Memory if the authorization is confirmed by CSEc.*

  - status\_t [CSEC\\_DRV\\_MPCompress](#) (const uint8\_t \*msg, uint16\_t msgLen, uint8\_t \*mpCompress, uint32\_t timeout)
- Compresses the given messages by accessing the Miyaguchi-Prenell compression feature with in the CSEc feature set.*

  - status\_t [CSEC\\_DRV\\_EncryptECBAsync](#) (csec\_key\_id\_t keyId, const uint8\_t \*plainText, uint32\_t length, uint8\_t \*cipherText)
- Asynchronously performs the AES-128 encryption in ECB mode.*

- status\_t [CSEC\\_DRV\\_DecryptECBAsync](#) (csec\_key\_id\_t keyId, const uint8\_t \*cipherText, uint32\_t length, uint8\_t \*plainText)  
*Asynchronously performs the AES-128 decryption in ECB mode.*
- status\_t [CSEC\\_DRV\\_EncryptCBCAsync](#) (csec\_key\_id\_t keyId, const uint8\_t \*plainText, uint32\_t length, const uint8\_t \*iv, uint8\_t \*cipherText)  
*Asynchronously performs the AES-128 encryption in CBC mode.*
- status\_t [CSEC\\_DRV\\_DecryptCBCAsync](#) (csec\_key\_id\_t keyId, const uint8\_t \*cipherText, uint32\_t length, const uint8\_t \*iv, uint8\_t \*plainText)  
*Asynchronously performs the AES-128 decryption in CBC mode.*
- status\_t [CSEC\\_DRV\\_GenerateMACAsync](#) (csec\_key\_id\_t keyId, const uint8\_t \*msg, uint32\_t msgLen, uint8\_t \*cmac)  
*Asynchronously calculates the MAC of a given message using CMAC with AES-128.*
- status\_t [CSEC\\_DRV\\_VerifyMACAsync](#) (csec\_key\_id\_t keyId, const uint8\_t \*msg, uint32\_t msgLen, const uint8\_t \*mac, uint16\_t macLen, bool \*verifStatus)  
*Asynchronously verifies the MAC of a given message using CMAC with AES-128.*
- status\_t [CSEC\\_DRV\\_GetAsyncCmdStatus](#) (void)  
*Checks the status of the execution of an asynchronous command.*
- void [CSEC\\_DRV\\_InstallCallback](#) (security\_callback\_t callbackFunc, void \*callbackParam)  
*Installs a callback function which will be invoked when an asynchronous command finishes its execution.*
- void [CSEC\\_DRV\\_CancelCommand](#) (void)  
*Cancels a previously launched asynchronous command.*

## 16.25.2 Data Structure Documentation

### 16.25.2.1 struct csec\_state\_t

Internal driver state information.

#### 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.

Implements : csec\_state\_t\_Class

Definition at line 183 of file csec\_driver.h.

#### Data Fields

- bool [cmdInProgress](#)
- [csec\\_cmd\\_t](#) cmd
- const uint8\_t \* [inputBuff](#)
- uint8\_t \* [outputBuff](#)
- uint32\_t [index](#)
- uint32\_t [fullSize](#)
- uint32\_t [partSize](#)
- [csec\\_key\\_id\\_t](#) keyId
- status\_t [errCode](#)
- const uint8\_t \* [iv](#)
- [csec\\_call\\_sequence\\_t](#) seq
- uint32\_t [msgLen](#)
- bool \* [verifStatus](#)
- bool [macWritten](#)
- const uint8\_t \* [mac](#)
- uint32\_t [macLen](#)
- security\_callback\_t [callback](#)
- void \* [callbackParam](#)

## Field Documentation

### 16.25.2.1.1 `security_callback_t callback`

The callback invoked when an asynchronous command is completed

Definition at line 200 of file `csec_driver.h`.

### 16.25.2.1.2 `void* callbackParam`

User parameter for the command completion callback

Definition at line 201 of file `csec_driver.h`.

### 16.25.2.1.3 `csec_cmd_t cmd`

Specifies the type of the command in execution

Definition at line 185 of file `csec_driver.h`.

### 16.25.2.1.4 `bool cmdInProgress`

Specifies if a command is in progress

Definition at line 184 of file `csec_driver.h`.

### 16.25.2.1.5 `status_t errCode`

Specifies the error code of the last executed command

Definition at line 192 of file `csec_driver.h`.

### 16.25.2.1.6 `uint32_t fullSize`

Specifies the size of the input of the command in execution

Definition at line 189 of file `csec_driver.h`.

### 16.25.2.1.7 `uint32_t index`

Specifies the index in the input buffer of the command in execution

Definition at line 188 of file `csec_driver.h`.

### 16.25.2.1.8 `const uint8_t* inputBuff`

Specifies the input of the command in execution

Definition at line 186 of file `csec_driver.h`.

### 16.25.2.1.9 `const uint8_t* iv`

Specifies the IV of the command in execution (for encryption/decryption using CBC mode)

Definition at line 193 of file `csec_driver.h`.

### 16.25.2.1.10 `csec_key_id_t keyId`

Specifies the key used for the command in execution

Definition at line 191 of file `csec_driver.h`.

### 16.25.2.1.11 `const uint8_t* mac`

Specifies the MAC to be verified for a MAC verification command

Definition at line 198 of file `csec_driver.h`.

**16.25.2.1.12 uint32\_t macLen**

Specifies the number of bits of the MAC to be verified for a MAC verification command

Definition at line 199 of file csec\_driver.h.

**16.25.2.1.13 bool macWritten**

Specifies if the MAC to be verified was written in CSE\_PRAM for a MAC verification command

Definition at line 197 of file csec\_driver.h.

**16.25.2.1.14 uint32\_t msgLen**

Specifies the message size (in bits) for the command in execution (for MAC generation/verification)

Definition at line 195 of file csec\_driver.h.

**16.25.2.1.15 uint8\_t\* outputBuff**

Specifies the output of the command in execution

Definition at line 187 of file csec\_driver.h.

**16.25.2.1.16 uint32\_t partSize**

Specifies the size of the chunk of the input currently processed

Definition at line 190 of file csec\_driver.h.

**16.25.2.1.17 csec\_call\_sequence\_t seq**

Specifies if the information is the first or a following function call.

Definition at line 194 of file csec\_driver.h.

**16.25.2.1.18 bool\* verifStatus**

Specifies the result of the last executed MAC verification command

Definition at line 196 of file csec\_driver.h.

**16.25.3 Macro Definition Documentation****16.25.3.1 #define CSEC\_STATUS\_BOOT\_FINISHED (0x8U)**

The bit is set when the secure booting has been finished by calling either CMD\_BOOT\_FAILURE or CMD\_BOOT\_OK or if CMD\_SECURE\_BOOT failed in verifying BOOT\_MAC.

Definition at line 70 of file csec\_driver.h.

**16.25.3.2 #define CSEC\_STATUS\_BOOT\_INIT (0x4U)**

The bit is set if the secure booting has been personalized during the boot sequence.

Definition at line 66 of file csec\_driver.h.

**16.25.3.3 #define CSEC\_STATUS\_BOOT\_OK (0x10U)**

The bit is set if the secure booting (CMD\_SECURE\_BOOT) succeeded. If CMD\_BOOT\_FAILURE is called the bit is erased.

Definition at line 73 of file csec\_driver.h.

#### 16.25.3.4 `#define CSEC_STATUS_BUSY (0x1U)`

The bit is set whenever SHE is processing a command.

Definition at line 61 of file `csec_driver.h`.

#### 16.25.3.5 `#define CSEC_STATUS_EXT_DEBUGGER (0x40U)`

The bit is set if an external debugger is connected to the chip.

Definition at line 77 of file `csec_driver.h`.

#### 16.25.3.6 `#define CSEC_STATUS_INT_DEBUGGER (0x80U)`

The bit is set if the internal debugging mechanisms of SHE are activated.

Definition at line 80 of file `csec_driver.h`.

#### 16.25.3.7 `#define CSEC_STATUS_RND_INIT (0x20U)`

The bit is set if the random number generator has been initialized.

Definition at line 75 of file `csec_driver.h`.

#### 16.25.3.8 `#define CSEC_STATUS_SECURE_BOOT (0x2U)`

The bit is set if the secure booting is activated.

Definition at line 63 of file `csec_driver.h`.

### 16.25.4 Typedef Documentation

#### 16.25.4.1 `typedef uint8_t csec_status_t`

Represents the status of the CSEc module. Provides one bit for each status code as per SHE specification. CSE↔  
C\_STATUS\_\* masks can be used for verifying the status.

Implements : `csec_status_t_Class`

Definition at line 89 of file `csec_driver.h`.

### 16.25.5 Enumeration Type Documentation

#### 16.25.5.1 `enum csec_boot_flavor_t`

Specifies the boot type for the BOOT\_DEFINE command.

Implements : `csec_boot_flavor_t_Class`

Enumerator

**`CSEC_BOOT_STRICT`**

**`CSEC_BOOT_SERIAL`**

**`CSEC_BOOT_PARALLEL`**

**`CSEC_BOOT_NOT_DEFINED`**

Definition at line 167 of file `csec_driver.h`.

#### 16.25.5.2 `enum csec_call_sequence_t`

Specifies if the information is the first or a following function call.

Implements : `csec_call_sequence_t_Class`

## Enumerator

***CSEC\_CALL\_SEQ\_FIRST***  
***CSEC\_CALL\_SEQ\_SUBSEQUENT***

Definition at line 157 of file csec\_driver.h.

## 16.25.5.3 enum csec\_cmd\_t

CSEc commands which follow the same values as the SHE command definition.

Implements : csec\_cmd\_t\_Class

## Enumerator

***CSEC\_CMD\_ENC\_ECB***  
***CSEC\_CMD\_ENC\_CBC***  
***CSEC\_CMD\_DEC\_ECB***  
***CSEC\_CMD\_DEC\_CBC***  
***CSEC\_CMD\_GENERATE\_MAC***  
***CSEC\_CMD\_VERIFY\_MAC***  
***CSEC\_CMD\_LOAD\_KEY***  
***CSEC\_CMD\_LOAD\_PLAIN\_KEY***  
***CSEC\_CMD\_EXPORT\_RAM\_KEY***  
***CSEC\_CMD\_INIT\_RNG***  
***CSEC\_CMD\_EXTEND\_SEED***  
***CSEC\_CMD\_RND***  
***CSEC\_CMD\_RESERVED\_1***  
***CSEC\_CMD\_BOOT\_FAILURE***  
***CSEC\_CMD\_BOOT\_OK***  
***CSEC\_CMD\_GET\_ID***  
***CSEC\_CMD\_BOOT\_DEFINE***  
***CSEC\_CMD\_DBG\_CHAL***  
***CSEC\_CMD\_DBG\_AUTH***  
***CSEC\_CMD\_RESERVED\_2***  
***CSEC\_CMD\_RESERVED\_3***  
***CSEC\_CMD\_MP\_COMPRESS***

Definition at line 127 of file csec\_driver.h.

## 16.25.5.4 enum csec\_key\_id\_t

Specify the KeyID to be used to implement the requested cryptographic operation.

Implements : csec\_key\_id\_t\_Class

## Enumerator

***CSEC\_SECRET\_KEY***  
***CSEC\_MASTER\_ECU***  
***CSEC\_BOOT\_MAC\_KEY***  
***CSEC\_BOOT\_MAC***  
***CSEC\_KEY\_1***

**CSEC\_KEY\_2**  
**CSEC\_KEY\_3**  
**CSEC\_KEY\_4**  
**CSEC\_KEY\_5**  
**CSEC\_KEY\_6**  
**CSEC\_KEY\_7**  
**CSEC\_KEY\_8**  
**CSEC\_KEY\_9**  
**CSEC\_KEY\_10**  
**CSEC\_RAM\_KEY**  
**CSEC\_KEY\_11**  
**CSEC\_KEY\_12**  
**CSEC\_KEY\_13**  
**CSEC\_KEY\_14**  
**CSEC\_KEY\_15**  
**CSEC\_KEY\_16**  
**CSEC\_KEY\_17**

Definition at line 97 of file csec\_driver.h.

#### 16.25.6 Function Documentation

##### 16.25.6.1 `status_t CSEC_DRV_BootDefine ( uint32_t bootSize, csec_boot_flavor_t bootFlavor )`

Implements an extension of the SHE standard to define both the user boot size and boot method.

The function implements an extension of the SHE standard to define both the user boot size and boot method.

##### Parameters

in	<i>bootSize</i>	Number of blocks of 128-bit data to check on boot. Maximum size is 512k↔ Bytes.
in	<i>bootFlavor</i>	The boot method.

##### Returns

Error Code after command execution.

Definition at line 928 of file csec\_driver.c.

##### 16.25.6.2 `status_t CSEC_DRV_BootFailure ( void )`

Signals a failure detected during later stages of the boot process.

The function is called during later stages of the boot process to detect a failure.

##### Returns

Error Code after command execution.

Definition at line 860 of file csec\_driver.c.



**16.25.6.3 status\_t CSEC\_DRV\_BootOK ( void )**

Marks a successful boot verification during later stages of the boot process.

The function is called during later stages of the boot process to mark successful boot verification.

**Returns**

Error Code after command execution.

Definition at line 894 of file csec\_driver.c.

**16.25.6.4 void CSEC\_DRV\_CancelCommand ( void )**

Cancels a previously launched asynchronous command.

Definition at line 1784 of file csec\_driver.c.

**16.25.6.5 status\_t CSEC\_DRV\_DbgAuth ( const uint8\_t \* *authorization* )**

Erases all keys (actual and outdated) stored in NVM Memory if the authorization is confirmed by CSEc.

This function erases all keys (actual and outdated) stored in NVM Memory if the authorization is confirmed by CSEc.

**Parameters**

<i>in</i>	<i>authorization</i>	Pointer to the 128-bit buffer containing the authorization value.
-----------	----------------------	---

**Returns**

Error Code after command execution.

Definition at line 1059 of file csec\_driver.c.

**16.25.6.6 status\_t CSEC\_DRV\_DbgChal ( uint8\_t \* *challenge* )**

Obtains a random number which the user shall use along with the MASTER\_ECU\_KEY and UID to return an authorization request.

This function obtains a random number which the user shall use along with the MASTER\_ECU\_KEY and UID to return an authorization request.

**Parameters**

<i>out</i>	<i>challenge</i>	Pointer to the 128-bit buffer where the challenge data will be stored.
------------	------------------	--

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 1019 of file csec\_driver.c.

**16.25.6.7 status\_t CSEC\_DRV\_DecryptCBC ( csec\_key\_id\_t *keyId*, const uint8\_t \* *cipherText*, uint32\_t *length*, const uint8\_t \* *iv*, uint8\_t \* *plainText*, uint32\_t *timeout* )**

Performs the AES-128 decryption in CBC mode.

This function performs the AES-128 decryption in CBC mode of the input cipher text buffer.

**Parameters**

in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>cipherText</i>	Pointer to the cipher text buffer.
in	<i>length</i>	Number of bytes of cipher text message to be decrypted. It should be multiple of 16 bytes.
in	<i>iv</i>	Pointer to the initialization vector buffer.
out	<i>plainText</i>	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.
in	<i>timeout</i>	Timeout in milliseconds.

#### Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 327 of file csec\_driver.c.

**16.25.6.8** `status_t CSEC_DRV_DecryptCBCAsync ( csec_key_id_t keyId, const uint8_t * cipherText, uint32_t length, const uint8_t * iv, uint8_t * plainText )`

Asynchronously performs the AES-128 decryption in CBC mode.

This function performs the AES-128 decryption in CBC mode of the input cipher text buffer, in an asynchronous manner.

#### Parameters

in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>cipherText</i>	Pointer to the cipher text buffer.
in	<i>length</i>	Number of bytes of cipher text message to be decrypted. It should be multiple of 16 bytes.
in	<i>iv</i>	Pointer to the initialization vector buffer.
out	<i>plainText</i>	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.

#### Returns

STATUS\_SUCCESS if the command was successfully launched, STATUS\_BUSY if another command was already launched. CSEC\_DRV\_GetAsyncCmdStatus can be used in order to check the execution status.

Definition at line 1282 of file csec\_driver.c.

**16.25.6.9** `status_t CSEC_DRV_DecryptECB ( csec_key_id_t keyId, const uint8_t * cipherText, uint32_t length, uint8_t * plainText, uint32_t timeout )`

Performs the AES-128 decryption in ECB mode.

This function performs the AES-128 decryption in ECB mode of the input cipher text buffer.

#### Parameters

in	<i>keyId</i>	KeyID used to perform the cryptographic operation
in	<i>cipherText</i>	Pointer to the cipher text buffer.
in	<i>length</i>	Number of bytes of cipher text message to be decrypted. It should be multiple of 16 bytes.
out	<i>plainText</i>	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.

in	<i>timeout</i>	Timeout in milliseconds.
----	----------------	--------------------------

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 219 of file csec\_driver.c.

**16.25.6.10** `status_t CSEC_DRV_DecryptECBAsync ( csec_key_id_t keyId, const uint8_t * cipherText, uint32_t length, uint8_t * plainText )`

Asynchronously performs the AES-128 decryption in ECB mode.

This function performs the AES-128 decryption in ECB mode of the input cipher text buffer, in an asynchronous manner.

**Parameters**

in	<i>keyId</i>	KeyID used to perform the cryptographic operation
in	<i>cipherText</i>	Pointer to the cipher text buffer.
in	<i>length</i>	Number of bytes of cipher text message to be decrypted. It should be multiple of 16 bytes.
out	<i>plainText</i>	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.

**Returns**

STATUS\_SUCCESS if the command was successfully launched, STATUS\_BUSY if another command was already launched. CSEC\_DRV\_GetAsyncCmdStatus can be used in order to check the execution status.

Definition at line 1215 of file csec\_driver.c.

**16.25.6.11** `void CSEC_DRV_Deinit ( void )`

Clears the internal state of the driver and disables the FTFC interrupt.

Definition at line 151 of file csec\_driver.c.

**16.25.6.12** `status_t CSEC_DRV_EncryptCBC ( csec_key_id_t keyId, const uint8_t * plainText, uint32_t length, const uint8_t * iv, uint8_t * cipherText, uint32_t timeout )`

Performs the AES-128 encryption in CBC mode.

This function performs the AES-128 encryption in CBC mode of the input plaintext buffer.

**Parameters**

in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>plainText</i>	Pointer to the plain text buffer.
in	<i>length</i>	Number of bytes of plain text message to be encrypted. It should be multiple of 16 bytes.
in	<i>iv</i>	Pointer to the initialization vector buffer.
in	<i>timeout</i>	Timeout in milliseconds.
out	<i>cipherText</i>	Pointer to the cipher text buffer. The buffer shall have the same size as the plain text buffer.
in	<i>timeout</i>	Timeout in milliseconds.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 271 of file csec\_driver.c.

**16.25.6.13** `status_t CSEC_DRV_EncryptCBCAsync ( csec_key_id_t keyId, const uint8_t * plainText, uint32_t length, const uint8_t * iv, uint8_t * cipherText )`

Asynchronously performs the AES-128 encryption in CBC mode.

This function performs the AES-128 encryption in CBC mode of the input plaintext buffer, in an asynchronous manner.

#### Parameters

in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>plainText</i>	Pointer to the plain text buffer.
in	<i>length</i>	Number of bytes of plain text message to be encrypted. It should be multiple of 16 bytes.
in	<i>iv</i>	Pointer to the initialization vector buffer.
out	<i>cipherText</i>	Pointer to the cipher text buffer. The buffer shall have the same size as the plain text buffer.

#### Returns

STATUS\_SUCCESS if the command was successfully launched, STATUS\_BUSY if another command was already launched. CSEC\_DRV\_GetAsyncCmdStatus can be used in order to check the execution status.

Definition at line 1247 of file csec\_driver.c.

**16.25.6.14** `status_t CSEC_DRV_EncryptECB ( csec_key_id_t keyId, const uint8_t * plainText, uint32_t length, uint8_t * cipherText, uint32_t timeout )`

Performs the AES-128 encryption in ECB mode.

This function performs the AES-128 encryption in ECB mode of the input plain text buffer

#### Parameters

in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>plainText</i>	Pointer to the plain text buffer.
in	<i>length</i>	Number of bytes of plain text message to be encrypted. It should be multiple of 16 bytes.
out	<i>cipherText</i>	Pointer to the cipher text buffer. The buffer shall have the same size as the plain text buffer.
in	<i>timeout</i>	Timeout in milliseconds.

#### Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 166 of file csec\_driver.c.

**16.25.6.15** `status_t CSEC_DRV_EncryptECBAsync ( csec_key_id_t keyId, const uint8_t * plainText, uint32_t length, uint8_t * cipherText )`

Asynchronously performs the AES-128 encryption in ECB mode.

This function performs the AES-128 encryption in ECB mode of the input plain text buffer, in an asynchronous manner.

#### Parameters

in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>plainText</i>	Pointer to the plain text buffer.
in	<i>length</i>	Number of bytes of plain text message to be encrypted. It should be multiple of 16 bytes.
out	<i>cipherText</i>	Pointer to the cipher text buffer. The buffer shall have the same size as the plain text buffer.

**Returns**

STATUS\_SUCCESS if the command was successfully launched, STATUS\_BUSY if another command was already launched. CSEC\_DRV\_GetAsyncCmdStatus can be used in order to check the execution status.

Definition at line 1183 of file csec\_driver.c.

**16.25.6.16** `status_t CSEC_DRV_ExportRAMKey ( uint8_t * m1, uint8_t * m2, uint8_t * m3, uint8_t * m4, uint8_t * m5 )`

Exports the RAM\_KEY into a format protected by SECRET\_KEY.

This function exports the RAM\_KEY into a format protected by SECRET\_KEY.

**Parameters**

out	<i>m1</i>	Pointer to a buffer where the M1 parameter will be exported.
out	<i>m2</i>	Pointer to a buffer where the M2 parameter will be exported.
out	<i>m3</i>	Pointer to a buffer where the M3 parameter will be exported.
out	<i>m4</i>	Pointer to a buffer where the M4 parameter will be exported.
out	<i>m5</i>	Pointer to a buffer where the M5 parameter will be exported.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 693 of file csec\_driver.c.

**16.25.6.17** `status_t CSEC_DRV_ExtendSeed ( const uint8_t * entropy )`

Extends the seed of the PRNG.

Extends the seed of the PRNG by compressing the former seed value and the supplied entropy into a new seed. This new seed is then to be used to generate a random number by invoking the CMD\_RND command. The random number generator must be initialized by CMD\_INIT\_RNG before the seed may be extended.

**Parameters**

in	<i>entropy</i>	Pointer to a 128-bit buffer containing the entropy.
----	----------------	---

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 782 of file csec\_driver.c.

**16.25.6.18** `status_t CSEC_DRV_GenerateMAC ( csec_key_id_t keyId, const uint8_t * msg, uint32_t msgLen, uint8_t * cmac, uint32_t timeout )`

Calculates the MAC of a given message using CMAC with AES-128.

This function calculates the MAC of a given message using CMAC with AES-128.

**Parameters**

in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>msg</i>	Pointer to the message buffer.
in	<i>msgLen</i>	Number of bits of message on which CMAC will be computed.
out	<i>cmac</i>	Pointer to the buffer containing the result of the CMAC computation.
in	<i>timeout</i>	Timeout in milliseconds.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 383 of file csec\_driver.c.

**16.25.6.19** `status_t CSEC_DRV_GenerateMACAddrMode ( csec_key_id_t keyId, const uint8_t * msg, uint32_t msgLen, uint8_t * cmac )`

Calculates the MAC of a given message (located in Flash) using CMAC with AES-128.

This function calculates the MAC of a given message using CMAC with AES-128. It is different from the CSEC\_DRV\_GenerateMAC function in the sense that it does not involve an extra copy of the data on which the CMAC is computed and the message pointer should be a pointer to Flash memory.

**Parameters**

in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>msg</i>	Pointer to the message buffer (pointing to Flash memory).
in	<i>msgLen</i>	Number of bits of message on which CMAC will be computed.
out	<i>cmac</i>	Pointer to the buffer containing the result of the CMAC computation.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 439 of file csec\_driver.c.

**16.25.6.20** `status_t CSEC_DRV_GenerateMACAsync ( csec_key_id_t keyId, const uint8_t * msg, uint32_t msgLen, uint8_t * cmac )`

Asynchronously calculates the MAC of a given message using CMAC with AES-128.

This function calculates the MAC of a given message using CMAC with AES-128, in an asynchronous manner.

**Parameters**

in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>msg</i>	Pointer to the message buffer.
in	<i>msgLen</i>	Number of bits of message on which CMAC will be computed.
out	<i>cmac</i>	Pointer to the buffer containing the result of the CMAC computation.

**Returns**

STATUS\_SUCCESS if the command was successfully launched, STATUS\_BUSY if another command was already launched. CSEC\_DRV\_GetAsyncCmdStatus can be used in order to check the execution status.

Definition at line 1317 of file csec\_driver.c.

**16.25.6.21** `status_t CSEC_DRV_GenerateRND ( uint8_t * rnd )`

Generates a vector of 128 random bits.

The function returns a vector of 128 random bits. The random number generator has to be initialized by calling CSEC\_DRV\_InitRNG before random numbers can be supplied.

**Parameters**

out	<i>rnd</i>	Pointer to a 128-bit buffer where the generated random number has to be stored.
-----	------------	---

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 820 of file csec\_driver.c.

**16.25.6.22 status\_t CSEC\_DRV\_GetAsyncCmdStatus ( void )**

Checks the status of the execution of an asynchronous command.

This function checks the status of the execution of an asynchronous command. If the command is still in progress, returns STATUS\_BUSY.

**Returns**

Error Code after command execution.

Definition at line 1390 of file csec\_driver.c.

**16.25.6.23 status\_t CSEC\_DRV\_GetID ( const uint8\_t \* challenge, uint8\_t \* uid, uint8\_t \* sreg, uint8\_t \* mac )**

Returns the identity (UID) and the value of the status register protected by a MAC over a challenge and the data.

This function returns the identity (UID) and the value of the status register protected by a MAC over a challenge and the data.

**Parameters**

in	<i>challenge</i>	Pointer to the 128-bit buffer containing Challenge data.
out	<i>uid</i>	Pointer to 120 bit buffer where the UID will be stored.
out	<i>sreg</i>	Value of the status register.
out	<i>mac</i>	Pointer to the 128 bit buffer where the MAC generated over challenge and UID and status will be stored.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 967 of file csec\_driver.c.

**16.25.6.24 static csec\_status\_t CSEC\_DRV\_GetStatus ( void ) [inline],[static]**

Returns the content of the status register.

The function shall return the content of the status register.

**Returns**

Value of the status register.

Implements : CSEC\_DRV\_GetStatus\_Activity

Definition at line 526 of file csec\_driver.h.

**16.25.6.25 void CSEC\_DRV\_Init ( csec\_state\_t \* state )**

Initializes the internal state of the driver and enables the FTFC interrupt.

**Parameters**

<i>in</i>	<i>state</i>	Pointer to the state structure which will be used for holding the internal state of the driver.
-----------	--------------	---

Definition at line 131 of file `csec_driver.c`.

**16.25.6.26** `status_t CSEC_DRV_InitRNG ( void )`

Initializes the seed and derives a key for the PRNG.

The function initializes the seed and derives a key for the PRNG. The function must be called before `CMD_RND` after every power cycle/reset.

**Returns**

Error Code after command execution.

Definition at line 745 of file `csec_driver.c`.

**16.25.6.27** `void CSEC_DRV_InstallCallback ( security_callback_t callbackFunc, void * callbackParam )`

Installs a callback function which will be invoked when an asynchronous command finishes its execution.

**Parameters**

<i>in</i>	<i>callbackFunc</i>	The function to be invoked.
<i>in</i>	<i>callbackParam</i>	The parameter to be passed to the callback function.

Definition at line 1769 of file `csec_driver.c`.

**16.25.6.28** `status_t CSEC_DRV_LoadKey ( csec_key_id_t keyId, const uint8_t * m1, const uint8_t * m2, const uint8_t * m3, uint8_t * m4, uint8_t * m5 )`

Updates an internal key per the SHE specification.

This function updates an internal key per the SHE specification.

**Parameters**

<i>in</i>	<i>keyId</i>	KeyID of the key to be updated.
<i>in</i>	<i>m1</i>	Pointer to the 128-bit M1 message containing the UID, Key ID and Authentication Key ID.
<i>in</i>	<i>m2</i>	Pointer to the 256-bit M2 message contains the new security flags, counter and the key value all encrypted using a derived key generated from the Authentication Key.
<i>in</i>	<i>m3</i>	Pointer to the 128-bit M3 message is a MAC generated over messages M1 and M2.
<i>out</i>	<i>m4</i>	Pointer to a 256 bits buffer where the computed M4 parameter is stored.
<i>out</i>	<i>m5</i>	Pointer to a 128 bits buffer where the computed M5 parameters is stored.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is `STATUS_SUCCESS`.

Definition at line 602 of file `csec_driver.c`.

**16.25.6.29** `status_t CSEC_DRV_LoadPlainKey ( const uint8_t * plainKey )`

Updates the RAM key memory slot with a 128-bit plaintext.

The function updates the RAM key memory slot with a 128-bit plaintext. The key is loaded without encryption and verification of the key, i.e. the key is handed over in plaintext. A plain key can only be loaded into the `RAM_KEY` slot.



**Parameters**

in	<i>plainKey</i>	Pointer to the 128-bit buffer containing the key that needs to be copied in R↔AM_KEY slot.
----	-----------------	--

**Returns**

Error Code after command execution.

Definition at line 656 of file csec\_driver.c.

**16.25.6.30** `status_t CSEC_DRV_MPCompress ( const uint8_t * msg, uint16_t msgLen, uint8_t * mpCompress, uint32_t timeout )`

Compresses the given messages by accessing the Miyaguchi-Prenell compression feature with in the CSEc feature set.

This function accesses a Miyaguchi-Prenell compression feature within the CSEc feature set to compress the given messages.

**Parameters**

in	<i>msg</i>	Pointer to the messages to be compressed. Messages must be pre-processed per SHE specification if they do not already meet the full 128-bit block size requirement.
in	<i>msgLen</i>	The number of 128 bit messages to be compressed.
out	<i>mpCompress</i>	Pointer to the 128 bit buffer storing the compressed data.
in	<i>timeout</i>	Timeout in milliseconds.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 1096 of file csec\_driver.c.

**16.25.6.31** `status_t CSEC_DRV_VerifyMAC ( csec_key_id_t keyId, const uint8_t * msg, uint32_t msgLen, const uint8_t * mac, uint16_t macLen, bool * verifStatus, uint32_t timeout )`

Verifies the MAC of a given message using CMAC with AES-128.

This function verifies the MAC of a given message using CMAC with AES-128.

**Parameters**

in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>msg</i>	Pointer to the message buffer.
in	<i>msgLen</i>	Number of bits of message on which CMAC will be computed.
in	<i>mac</i>	Pointer to the buffer containing the CMAC to be verified.
in	<i>macLen</i>	Number of bits of the CMAC to be compared. A macLength value of zero indicates that all 128-bits are compared.
out	<i>verifStatus</i>	Status of MAC verification command (true: verification operation passed, false: verification operation failed).
in	<i>timeout</i>	Timeout in milliseconds.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 484 of file csec\_driver.c.

**16.25.6.32** `status_t CSEC_DRV_VerifyMACAddrMode ( csec_key_id_t keyId, const uint8_t * msg, uint32_t msgLen, const uint8_t * mac, uint16_t macLen, bool * verifStatus )`

Verifies the MAC of a given message (located in Flash) using CMAC with AES-128.

This function verifies the MAC of a given message using CMAC with AES-128. It is different from the `CSEC_DRV_VerifyMAC` function in the sense that it does not involve an extra copy of the data on which the CMAC is computed and the message pointer should be a pointer to Flash memory.

#### Parameters

in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>msg</i>	Pointer to the message buffer (pointing to Flash memory).
in	<i>msgLen</i>	Number of bits of message on which CMAC will be computed.
in	<i>mac</i>	Pointer to the buffer containing the CMAC to be verified.
in	<i>macLen</i>	Number of bits of the CMAC to be compared. A <i>macLength</i> value of zero indicates that all 128-bits are compared.
out	<i>verifStatus</i>	Status of MAC verification command (true: verification operation passed, false: verification operation failed).

#### Returns

Error Code after command execution. Output parameters are valid if the error code is `STATUS_SUCCESS`.

Definition at line 547 of file `csec_driver.c`.

**16.25.6.33** `status_t CSEC_DRV_VerifyMACAsync ( csec_key_id_t keyId, const uint8_t * msg, uint32_t msgLen, const uint8_t * mac, uint16_t macLen, bool * verifStatus )`

Asynchronously verifies the MAC of a given message using CMAC with AES-128.

This function verifies the MAC of a given message using CMAC with AES-128, in an asynchronous manner.

#### Parameters

in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>msg</i>	Pointer to the message buffer.
in	<i>msgLen</i>	Number of bits of message on which CMAC will be computed.
in	<i>mac</i>	Pointer to the buffer containing the CMAC to be verified.
in	<i>macLen</i>	Number of bits of the CMAC to be compared. A <i>macLength</i> value of zero indicates that all 128-bits are compared.
out	<i>verifStatus</i>	Status of MAC verification command (true: verification operation passed, false: verification operation failed).

#### Returns

`STATUS_SUCCESS` if the command was successfully launched, `STATUS_BUSY` if another command was already launched. `CSEC_DRV_GetAsyncCmdStatus` can be used in order to check the execution status.

Definition at line 1350 of file `csec_driver.c`.

## 16.26 Clock

### 16.26.1 Detailed Description

#### Dynamic clock setting

- status\_t [CLOCK\\_DRV\\_GetFreq](#) (clock\_names\_t clockName, uint32\_t \*frequency)  
*Gets the clock frequency for a specific clock name.*
- status\_t [CLOCK\\_DRV\\_Init](#) (clock\_user\_config\_t const \*config)  
*Set clock configuration according to pre-defined structure.*

### 16.26.2 Function Documentation

#### 16.26.2.1 status\_t CLOCK\_DRV\_GetFreq ( clock\_names\_t clockName, uint32\_t \* frequency )

Gets the clock frequency for a specific clock name.

This function checks the current clock configurations and then calculates the clock frequency for a specific clock name defined in clock\_names\_t. Clock modules must be properly configured before using this function. See features.h 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 STATUS\_UNSUPPORTED. If frequency is required for a peripheral and the module is not clocked, then STATUS\_MCU\_GATED\_OFF status is returned. Frequency is returned if a valid address is provided. If frequency is required for a peripheral that doesn't support protocol clock, the zero value is provided.

##### Parameters

in	clockName	Clock names defined in clock_names_t
out	frequency	Returned clock frequency value in Hertz

##### Returns

status Error code defined in status\_t

Definition at line 1902 of file clock\_S32K1xx.c.

#### 16.26.2.2 status\_t CLOCK\_DRV\_Init ( clock\_user\_config\_t const \* config )

Set clock configuration according to pre-defined structure.

This function sets system to target clock configuration; It sets the clock modules registers for clock mode change.

##### Parameters

in	config	Pointer to configuration structure.
----	--------	-------------------------------------

##### 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 correctly.

If the configuration structure is NULL, the function will set a default configuration for clock.

Definition at line 603 of file clock\_S32K1xx.c.

## 16.27 Clock Manager

### 16.27.1 Detailed Description

This module covers the clock management API and clock related functionality.

This section describes the programming interface of the clock\_manager driver. Clock\_manager achieves its functionality by configuring the hardware modules involved in clock distribution and management.

#### Driver consideration

The Clock Manager driver is developed on top of an appropriate hardware access layer. The Clock Manager provides API to handle the clock configuration. The Driver uses structures for configuration. The actual format of the structure is defined by the underlying device specific header file. These structures may be generated using S32DS configuration. The user application can use the default for most settings, changing only what is necessary.

This driver provides functions for initializing system clock and peripheral clock.

All methods that access the hardware layer will return an error code to signal if the operation succeeded or failed. The values are defined by the status\_t enumeration, and the possible values include: success, error.

#### Modules

- [Clock Manager Driver](#)

*This module covers the device-specific clock\_manager functionality implemented for S32K1xx SOC.*

## 16.28 Clock Manager Driver

### 16.28.1 Detailed Description

This module covers the device-specific clock\_manager functionality implemented for S32K1xx SOC.

The support for S32K1xx consist in the following items:

1. Clock names enumeration clock\_names\_t is an enumeration which contains all clock names which are relevant for S32K1xx.
2. Submodule configuration structures
  - [scg\\_config\\_t](#)
  - [pcc\\_config\\_t](#)
  - [sim\\_clock\\_config\\_t](#)
  - [pmc\\_config\\_t](#)
3. Submodule configuration functions The following functions were implemented for S32K1xx:
  - CLOCK\_SYS\_SetScgConfiguration
  - CLOCK\_SYS\_SetPccConfiguration
  - CLOCK\_SYS\_SetSimConfiguration
  - CLOCK\_SYS\_SetPmcConfiguration

#### Hardware background

Features of clock\_manager module include the following clock sources:

- 4 - 40 MHz fast external oscillator (SOSC)
- 48 MHz Fast Internal RC oscillator (FIRC)
- 8 MHz Slow Internal RC oscillator (SIRC)
- 128 kHz Low Power Oscillator (LPO)
- Up to 112 MHz (HSRUN) System Phased Lock Loop (SPLL)

#### How to use the CLOCK\_MANAGER driver in your application

In order to be able to use the clock\_manager in your application, **CLOCK\_DRV\_Init** function has to be called. The same function is called when another configuration is loaded and clock configuration is updated.

#### Code Example

This is an example for switching between two configurations:

```
CLOCK_SYS_Init(g_clockManConfigsArr,  
              CLOCK_MANAGER_CONFIG_CNT,  
              g_clockManCallbacksArr,  
              CLOCK_MANAGER_CALLBACK_CNT);  
  
CLOCK_SYS_UpdateConfiguration(0,  
                             CLOCK_MANAGER_POLICY_FORCIBLE);  
CLOCK_SYS_UpdateConfiguration(1,  
                             CLOCK_MANAGER_POLICY_FORCIBLE);
```

## Notes

Current implementation assumes that the clock configurations are valid and are applied in a valid sequence. Mainly this means that the configuration doesn't reinitialize the clock used as the system clock.

According to Errata e10777, when the power mode is being switched, the core clock frequency is incorrectly read. As a result, when switching from VLPR mode to HSRUN, the frequency has to be read twice or after some time has passed.

The S32DS do not support generate Callbacks configuration. It's alway empty.

## Integration guideline

### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\drivers\src\clock\S32K1xx\clock_S32K1xx.c
```

### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc\  
${S32SDK_PATH}\platform\drivers\src\clock\  
${S32SDK_PATH}\platform\drivers\src\clock\S32K1xx\
```

### Compile symbols

No special symbols are required for this component

### Dependencies

### [Interrupt Manager \(Interrupt\)](#)

## Data Structures

- struct [sim\\_clock\\_out\\_config\\_t](#)  
*SIM ClockOut configuration. Implements sim\_clock\_out\_config\_t\_Class. [More...](#)*
- struct [sim\\_lpo\\_clock\\_config\\_t](#)  
*SIM LPO Clocks configuration. Implements sim\_lpo\_clock\_config\_t\_Class. [More...](#)*
- struct [sim\\_tclk\\_config\\_t](#)  
*SIM Platform Gate Clock configuration. Implements sim\_tclk\_config\_t\_Class. [More...](#)*
- struct [sim\\_plat\\_gate\\_config\\_t](#)  
*SIM Platform Gate Clock configuration. Implements sim\_plat\_gate\_config\_t\_Class. [More...](#)*
- struct [sim\\_qspi\\_ref\\_clk\\_gating\\_t](#)  
*SIM QSPI reference clock gating. Implements sim\_qspi\_ref\_clk\_gating\_t\_Class. [More...](#)*
- struct [sim\\_trace\\_clock\\_config\\_t](#)  
*SIM Debug Trace clock configuration. Implements sim\_trace\_clock\_config\_t\_Class. [More...](#)*
- struct [sim\\_clock\\_config\\_t](#)  
*SIM configure structure. Implements sim\_clock\_config\_t\_Class. [More...](#)*
- struct [scg\\_system\\_clock\\_config\\_t](#)  
*SCG system clock configuration. Implements scg\_system\_clock\_config\_t\_Class. [More...](#)*
- struct [scg\\_sosc\\_config\\_t](#)  
*SCG system OSC configuration. Implements scg\_sosc\_config\_t\_Class. [More...](#)*
- struct [scg\\_sirc\\_config\\_t](#)  
*SCG slow IRC clock configuration. Implements scg\_sirc\_config\_t\_Class. [More...](#)*
- struct [scg\\_firc\\_config\\_t](#)

- SCG fast IRC clock configuration. Implements `scg_firc_config_t` Class. [More...](#)*
- struct `scg_spll_config_t`
  - SCG system PLL configuration. Implements `scg_spll_config_t` Class. [More...](#)*
- struct `scg_rtc_config_t`
  - SCG RTC configuration. Implements `scg_rtc_config_t` Class. [More...](#)*
- struct `scg_clock_mode_config_t`
  - SCG Clock Mode Configuration structure. Implements `scg_clock_mode_config_t` Class. [More...](#)*
- struct `scg_clockout_config_t`
  - SCG ClockOut Configuration structure. Implements `scg_clockout_config_t` Class. [More...](#)*
- struct `scg_config_t`
  - SCG configure structure. Implements `scg_config_t` Class. [More...](#)*
- struct `peripheral_clock_config_t`
  - PCC peripheral instance clock configuration. Implements `peripheral_clock_config_t` Class. [More...](#)*
- struct `pcc_config_t`
  - PCC configuration. Implements `pcc_config_t` Class. [More...](#)*
- struct `pmc_lpo_clock_config_t`
  - PMC LPO configuration. Implements `pmc_lpo_clock_config_t` Class. [More...](#)*
- struct `pmc_config_t`
  - PMC configure structure. Implements `pmc_config_t` Class. [More...](#)*
- struct `clock_manager_user_config_t`
  - Clock configuration structure. Implements `clock_manager_user_config_t` Class. [More...](#)*
- struct `module_clk_config_t`
  - module clock configuration. Implements `module_clk_config_t` Class [More...](#)*
- struct `sys_clk_config_t`
  - System clock configuration. Implements `sys_clk_config_t` Class. [More...](#)*
- struct `clock_source_config_t`
  - Clock source configuration. Implements `clock_source_config_t` Class. [More...](#)*
- struct `clock_notify_struct_t`
  - Clock notification structure passed to clock callback function. Implements `clock_notify_struct_t` Class. [More...](#)*
- struct `clock_manager_callback_user_config_t`
  - Structure for callback function and its parameter. Implements `clock_manager_callback_user_config_t` Class. [More...](#)*
- struct `clock_manager_state_t`
  - Clock manager state structure. Implements `clock_manager_state_t` Class. [More...](#)*

## Macros

- #define `NUMBER_OF_TCLK_INPUTS` 3U
  - TClk clock frequency.*
- #define `SYS_CLK_MAX_NO` 3U
  - The maximum number of system clock dividers and system clock divider indexes.*
- #define `CORE_CLK_INDEX` 0U
- #define `BUS_CLK_INDEX` 1U
- #define `SLOW_CLK_INDEX` 2U
- #define `CLK_SRC_OFF` 0x00U
- #define `CLK_SRC_SOSC` 0x01U
- #define `CLK_SRC_SIRC` 0x02U
- #define `CLK_SRC_FIRC` 0x03U
- #define `CLK_SRC_SPLL` 0x06U
- #define `CLK_SRC_SOSC_DIV1` 0x01U
- #define `CLK_SRC_SIRC_DIV1` 0x02U
- #define `CLK_SRC_FIRC_DIV1` 0x03U

- #define CLK\_SRC\_SPLL\_DIV1 0x06U
- #define CLK\_SRC\_SOSC\_DIV2 0x01U
- #define CLK\_SRC\_SIRC\_DIV2 0x02U
- #define CLK\_SRC\_FIRC\_DIV2 0x03U
- #define CLK\_SRC\_SPLL\_DIV2 0x06U

## Typedefs

- typedef uint8\_t peripheral\_clock\_source\_t  
*PCC clock source select Implements peripheral\_clock\_source\_t Class.*
- typedef clock\_manager\_user\_config\_t clock\_user\_config\_t
- typedef status\_t(\* clock\_manager\_callback\_t) (clock\_notify\_struct\_t \*notify, void \*callbackData)  
*Type of clock callback functions.*

## Enumerations

- enum sim\_rtc\_clk\_sel\_src\_t { SIM\_RTCCLK\_SEL\_SOSCDIV1\_CLK = 0x0U, SIM\_RTCCLK\_SEL\_LPO\_32K = 0x1U, SIM\_RTCCLK\_SEL\_RTC\_CLKIN = 0x2U, SIM\_RTCCLK\_SEL\_FIRCDIV1\_CLK = 0x3U }  
*SIM CLK32KSEL clock source select Implements sim\_rtc\_clk\_sel\_src\_t Class.*
- enum sim\_lpclk\_sel\_src\_t { SIM\_LPO\_CLK\_SEL\_LPO\_128K = 0x0, SIM\_LPO\_CLK\_SEL\_NO\_CLOCK = 0x1, SIM\_LPO\_CLK\_SEL\_LPO\_32K = 0x2, SIM\_LPO\_CLK\_SEL\_LPO\_1K = 0x3 }  
*SIM LPOCLKSEL clock source select Implements sim\_lpclk\_sel\_src\_t Class.*
- enum sim\_clkout\_src\_t {  
SIM\_CLKOUT\_SEL\_SYSTEM\_SCG\_CLKOUT = 0U, SIM\_CLKOUT\_SEL\_SYSTEM\_SOSC\_DIV2\_CLK = 2U, SIM\_CLKOUT\_SEL\_SYSTEM\_SIRC\_DIV2\_CLK = 4U, SIM\_CLKOUT\_SEL\_SYSTEM\_FIRC\_DIV2\_CLK = 6U,  
SIM\_CLKOUT\_SEL\_SYSTEM\_HCLK = 7U, SIM\_CLKOUT\_SEL\_SYSTEM\_SPLL\_DIV2\_CLK = 8U, SIM\_CLKOUT\_SEL\_SYSTEM\_BUS\_CLK = 9U, SIM\_CLKOUT\_SEL\_SYSTEM\_LPO\_128K\_CLK = 10U,  
SIM\_CLKOUT\_SEL\_SYSTEM\_LPO\_CLK = 12U, SIM\_CLKOUT\_SEL\_SYSTEM\_RTC\_CLK = 14U }  
*SIM CLKOUT select Implements sim\_clkout\_src\_t Class.*
- enum sim\_clkout\_div\_t {  
SIM\_CLKOUT\_DIV\_BY\_1 = 0x0U, SIM\_CLKOUT\_DIV\_BY\_2 = 0x1U, SIM\_CLKOUT\_DIV\_BY\_3 = 0x2U, SIM\_CLKOUT\_DIV\_BY\_4 = 0x3U,  
SIM\_CLKOUT\_DIV\_BY\_5 = 0x4U, SIM\_CLKOUT\_DIV\_BY\_6 = 0x5U, SIM\_CLKOUT\_DIV\_BY\_7 = 0x6U, SIM\_CLKOUT\_DIV\_BY\_8 = 0x7U }  
*SIM CLKOUT divider Implements sim\_clkout\_div\_t Class.*
- enum clock\_trace\_src\_t { CLOCK\_TRACE\_SRC\_CORE\_CLK = 0x0 }  
*Debug trace clock source select Implements clock\_trace\_src\_t Class.*
- enum scg\_system\_clock\_src\_t { SCG\_SYSTEM\_CLOCK\_SRC\_SYS\_OSC = 1U, SCG\_SYSTEM\_CLOCK\_SRC\_SIRC = 2U, SCG\_SYSTEM\_CLOCK\_SRC\_FIRC = 3U, SCG\_SYSTEM\_CLOCK\_SRC\_NONE = 255U }  
*SCG system clock source. Implements scg\_system\_clock\_src\_t Class.*
- enum scg\_system\_clock\_div\_t {  
SCG\_SYSTEM\_CLOCK\_DIV\_BY\_1 = 0U, SCG\_SYSTEM\_CLOCK\_DIV\_BY\_2 = 1U, SCG\_SYSTEM\_CLOCK\_DIV\_BY\_3 = 2U, SCG\_SYSTEM\_CLOCK\_DIV\_BY\_4 = 3U,  
SCG\_SYSTEM\_CLOCK\_DIV\_BY\_5 = 4U, SCG\_SYSTEM\_CLOCK\_DIV\_BY\_6 = 5U, SCG\_SYSTEM\_CLOCK\_DIV\_BY\_7 = 6U, SCG\_SYSTEM\_CLOCK\_DIV\_BY\_8 = 7U,  
SCG\_SYSTEM\_CLOCK\_DIV\_BY\_9 = 8U, SCG\_SYSTEM\_CLOCK\_DIV\_BY\_10 = 9U, SCG\_SYSTEM\_CLOCK\_DIV\_BY\_11 = 10U, SCG\_SYSTEM\_CLOCK\_DIV\_BY\_12 = 11U,  
SCG\_SYSTEM\_CLOCK\_DIV\_BY\_13 = 12U, SCG\_SYSTEM\_CLOCK\_DIV\_BY\_14 = 13U, SCG\_SYSTEM\_CLOCK\_DIV\_BY\_15 = 14U, SCG\_SYSTEM\_CLOCK\_DIV\_BY\_16 = 15U }  
*SCG system clock divider value. Implements scg\_system\_clock\_div\_t Class.*



- enum `scg_async_clock_div_t` {  
`SCG_ASYNC_CLOCK_DISABLE` = 0U, `SCG_ASYNC_CLOCK_DIV_BY_1` = 1U, `SCG_ASYNC_CLOCK_DIV_BY_2` = 2U, `SCG_ASYNC_CLOCK_DIV_BY_4` = 3U,  
`SCG_ASYNC_CLOCK_DIV_BY_8` = 4U, `SCG_ASYNC_CLOCK_DIV_BY_16` = 5U, `SCG_ASYNC_CLOCK_DIV_BY_32` = 6U, `SCG_ASYNC_CLOCK_DIV_BY_64` = 7U }  
*SCG asynchronous clock divider value. Implements `scg_async_clock_div_t` Class.*
- enum `scg_sosc_monitor_mode_t` { `SCG_SOSC_MONITOR_DISABLE` = 0U, `SCG_SOSC_MONITOR_INT` = 1U, `SCG_SOSC_MONITOR_RESET` = 2U }  
*SCG system OSC monitor mode. Implements `scg_sosc_monitor_mode_t` Class.*
- enum `scg_sosc_range_t` { `SCG_SOSC_RANGE_MID` = 2U, `SCG_SOSC_RANGE_HIGH` = 3U }  
*SCG OSC frequency range select Implements `scg_sosc_range_t` Class.*
- enum `scg_sosc_gain_t` { `SCG_SOSC_GAIN_LOW` = 0x0, `SCG_SOSC_GAIN_HIGH` = 0x1 }  
*SCG OSC high gain oscillator select. Implements `scg_sosc_gain_t` Class.*
- enum `scg_sosc_ext_ref_t` { `SCG_SOSC_REF_EXT` = 0x0, `SCG_SOSC_REF_OSC` = 0x1 }  
*SCG OSC external reference clock select. Implements `scg_sosc_ext_ref_t` Class.*
- enum `scg_sirc_range_t` { `SCG_SIRC_RANGE_HIGH` = 1U }  
*SCG slow IRC clock frequency range. Implements `scg_sirc_range_t` Class.*
- enum `scg_firc_range_t` { `SCG_FIRC_RANGE_48M` }  
*SCG fast IRC clock frequency range. Implements `scg_firc_range_t` Class.*
- enum `scg_sppl_monitor_mode_t` { `SCG_SPPL_MONITOR_DISABLE` = 0U, `SCG_SPPL_MONITOR_INT` = 1U, `SCG_SPPL_MONITOR_RESET` = 2U }  
*SCG system PLL monitor mode. Implements `scg_sppl_monitor_mode_t` Class.*
- enum `scg_sppl_clock_prediv_t` {  
`SCG_SPPL_CLOCK_PREDIV_BY_1` = 0U, `SCG_SPPL_CLOCK_PREDIV_BY_2` = 1U, `SCG_SPPL_CLOCK_PREDIV_BY_3` = 2U, `SCG_SPPL_CLOCK_PREDIV_BY_4` = 3U,  
`SCG_SPPL_CLOCK_PREDIV_BY_5` = 4U, `SCG_SPPL_CLOCK_PREDIV_BY_6` = 5U, `SCG_SPPL_CLOCK_PREDIV_BY_7` = 6U, `SCG_SPPL_CLOCK_PREDIV_BY_8` = 7U }  
*SCG system PLL predivider.*
- enum `scg_sppl_clock_multiply_t` {  
`SCG_SPPL_CLOCK_MULTIPLY_BY_16` = 0U, `SCG_SPPL_CLOCK_MULTIPLY_BY_17` = 1U, `SCG_SPPL_CLOCK_MULTIPLY_BY_18` = 2U, `SCG_SPPL_CLOCK_MULTIPLY_BY_19` = 3U,  
`SCG_SPPL_CLOCK_MULTIPLY_BY_20` = 4U, `SCG_SPPL_CLOCK_MULTIPLY_BY_21` = 5U, `SCG_SPPL_CLOCK_MULTIPLY_BY_22` = 6U, `SCG_SPPL_CLOCK_MULTIPLY_BY_23` = 7U,  
`SCG_SPPL_CLOCK_MULTIPLY_BY_24` = 8U, `SCG_SPPL_CLOCK_MULTIPLY_BY_25` = 9U, `SCG_SPPL_CLOCK_MULTIPLY_BY_26` = 10U, `SCG_SPPL_CLOCK_MULTIPLY_BY_27` = 11U,  
`SCG_SPPL_CLOCK_MULTIPLY_BY_28` = 12U, `SCG_SPPL_CLOCK_MULTIPLY_BY_29` = 13U, `SCG_SPPL_CLOCK_MULTIPLY_BY_30` = 14U, `SCG_SPPL_CLOCK_MULTIPLY_BY_31` = 15U,  
`SCG_SPPL_CLOCK_MULTIPLY_BY_32` = 16U, `SCG_SPPL_CLOCK_MULTIPLY_BY_33` = 17U, `SCG_SPPL_CLOCK_MULTIPLY_BY_34` = 18U, `SCG_SPPL_CLOCK_MULTIPLY_BY_35` = 19U,  
`SCG_SPPL_CLOCK_MULTIPLY_BY_36` = 20U, `SCG_SPPL_CLOCK_MULTIPLY_BY_37` = 21U, `SCG_SPPL_CLOCK_MULTIPLY_BY_38` = 22U, `SCG_SPPL_CLOCK_MULTIPLY_BY_39` = 23U,  
`SCG_SPPL_CLOCK_MULTIPLY_BY_40` = 24U, `SCG_SPPL_CLOCK_MULTIPLY_BY_41` = 25U, `SCG_SPPL_CLOCK_MULTIPLY_BY_42` = 26U, `SCG_SPPL_CLOCK_MULTIPLY_BY_43` = 27U,  
`SCG_SPPL_CLOCK_MULTIPLY_BY_44` = 28U, `SCG_SPPL_CLOCK_MULTIPLY_BY_45` = 29U, `SCG_SPPL_CLOCK_MULTIPLY_BY_46` = 30U, `SCG_SPPL_CLOCK_MULTIPLY_BY_47` = 31U }  
*SCG system PLL multiplier.*
- enum `peripheral_clock_frac_t` { `MULTIPLY_BY_ONE` = 0x00U, `MULTIPLY_BY_TWO` = 0x01U }  
*PCC fractional value select Implements `peripheral_clock_frac_t` Class.*
- enum `peripheral_clock_divider_t` {  
`DIVIDE_BY_ONE` = 0x00U, `DIVIDE_BY_TWO` = 0x01U, `DIVIDE_BY_THREE` = 0x02U, `DIVIDE_BY_FOUR` = 0x03U,  
`DIVIDE_BY_FIVE` = 0x04U, `DIVIDE_BY_SIX` = 0x05U, `DIVIDE_BY_SEVEN` = 0x06U, `DIVIDE_BY_EIGHTH` = 0x07U }  
*PCC divider value select Implements `peripheral_clock_divider_t` Class.*

- enum `pwr_modes_t` {  
`NO_MODE` = 0U, `RUN_MODE` = (1U<<0U), `VLPR_MODE` = (1U<<1U), `HSRUN_MODE` = (1U<<2U),  
`STOP_MODE` = (1U<<3U), `VLPS_MODE` = (1U<<4U), `ALL_MODES` = 0x7FFFFFFF }  
*Power mode. Implements `pwr_modes_t` Class.*
- enum `xosc_ref_t` { `XOSC_EXT_REF` = 0U, `XOSC_INT_OSC` = 1U }  
*XOSC reference clock select (internal oscillator is bypassed or not) Implements `xosc_ref_t` Class.*
- enum `clock_manager_notify_t` { `CLOCK_MANAGER_NOTIFY_RECOVER` = 0x00U, `CLOCK_MANAGER_NOTIFY_BEFORE` = 0x01U, `CLOCK_MANAGER_NOTIFY_AFTER` = 0x02U }  
*The clock notification type. Implements `clock_manager_notify_t` Class.*
- enum `clock_manager_callback_type_t` { `CLOCK_MANAGER_CALLBACK_BEFORE` = 0x01U, `CLOCK_MANAGER_CALLBACK_AFTER` = 0x02U, `CLOCK_MANAGER_CALLBACK_BEFORE_AFTER` = 0x03U }  
*The callback type, indicates what kinds of notification this callback handles. Implements `clock_manager_callback_type_t` Class.*
- enum `clock_manager_policy_t` { `CLOCK_MANAGER_POLICY_AGREEMENT`, `CLOCK_MANAGER_POLICY_FORCIBLE` }  
*Clock transition policy. Implements `clock_manager_policy_t` Class.*

## Functions

- void `CLOCK_DRV_SetModuleClock` (`clock_names_t` peripheralClock, const `module_clk_config_t` \*moduleClkConfig)  
*Configures module clock.*
- status\_t `CLOCK_DRV_SetSystemClock` (const `pwr_modes_t` \*mode, const `sys_clk_config_t` \*sysClkConfig)  
*Configures the system clocks.*
- void `CLOCK_DRV_GetSystemClockSource` (`sys_clk_config_t` \*sysClkConfig)  
*Gets the system clock source.*
- status\_t `CLOCK_DRV_SetClockSource` (`clock_names_t` clockSource, const `clock_source_config_t` \*clkSrcConfig)  
*This function configures a clock source.*
- status\_t `CLOCK_SYS_Init` (`clock_manager_user_config_t` const \*\*clockConfigsPtr, uint8\_t configsNumber, `clock_manager_callback_user_config_t` \*\*callbacksPtr, uint8\_t callbacksNumber)  
*Install pre-defined clock configurations.*
- status\_t `CLOCK_SYS_UpdateConfiguration` (uint8\_t targetConfigIndex, `clock_manager_policy_t` policy)  
*Set system clock configuration according to pre-defined structure.*
- status\_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.*
- status\_t `CLOCK_SYS_GetFreq` (`clock_names_t` clockName, uint32\_t \*frequency)  
*Wrapper over `CLOCK_DRV_GetFreq` function. It's part of the old API.*

## Variables

- const uint8\_t `peripheralFeaturesList` [CLOCK\_NAME\_COUNT]  
*Peripheral features list Constant array storing the mappings between clock names of the peripherals and feature lists.*
- uint32\_t `g_TClkFreq` [NUMBER\_OF\_TCLK\_INPUTS]
- uint32\_t `g_xtal0ClkFreq`  
*EXTAL0 clock frequency.*
- uint32\_t `g_RtcClkInFreq`  
*RTC\_CLKIN clock frequency.*

## SCG Clockout.

- enum `scg_clockout_src_t` {  
`SCG_CLOCKOUT_SRC_SCG_SLOW` = 0U, `SCG_CLOCKOUT_SRC_SOSC` = 1U, `SCG_CLOCKOUT_SRC_SIRC` = 2U, `SCG_CLOCKOUT_SRC_FIRC` = 3U,  
`SCG_CLOCKOUT_SRC_SPLL` = 6U }

*SCG ClockOut type. Implements `scg_clockout_src_t_Class`.*

## 16.28.2 Data Structure Documentation

16.28.2.1 struct `sim_clock_out_config_t`

SIM ClockOut configuration. Implements `sim_clock_out_config_t_Class`.

Definition at line 142 of file `clock_S32K1xx.h`.

## Data Fields

- bool `initialize`
- bool `enable`
- `sim_clkout_src_t` `source`
- `sim_clkout_div_t` `divider`

## Field Documentation

16.28.2.1.1 `sim_clkout_div_t` `divider`

SIM ClockOut divide ratio.

Definition at line 147 of file `clock_S32K1xx.h`.

16.28.2.1.2 bool `enable`

SIM ClockOut enable.

Definition at line 145 of file `clock_S32K1xx.h`.

16.28.2.1.3 bool `initialize`

Initialize or not the ClockOut clock.

Definition at line 144 of file `clock_S32K1xx.h`.

16.28.2.1.4 `sim_clkout_src_t` `source`

SIM ClockOut source select.

Definition at line 146 of file `clock_S32K1xx.h`.

16.28.2.2 struct `sim_lpo_clock_config_t`

SIM LPO Clocks configuration. Implements `sim_lpo_clock_config_t_Class`.

Definition at line 155 of file `clock_S32K1xx.h`.

## Data Fields

- bool `initialize`
- `sim_rtc_clk_sel_src_t` `sourceRtcClk`
- `sim_lpoclk_sel_src_t` `sourceLpoClk`
- bool `enableLpo32k`
- bool `enableLpo1k`

## Field Documentation

### 16.28.2.2.1 bool enableLpo1k

MSCM Clock Gating Control enable.

Definition at line 161 of file clock\_S32K1xx.h.

### 16.28.2.2.2 bool enableLpo32k

MSCM Clock Gating Control enable.

Definition at line 160 of file clock\_S32K1xx.h.

### 16.28.2.2.3 bool initialize

Initialize or not the LPO clock.

Definition at line 157 of file clock\_S32K1xx.h.

### 16.28.2.2.4 sim\_lpoclk\_sel\_src\_t sourceLpoClk

LPO clock source select.

Definition at line 159 of file clock\_S32K1xx.h.

### 16.28.2.2.5 sim\_rtc\_clk\_sel\_src\_t sourceRtcClk

RTC\_CLK source select.

Definition at line 158 of file clock\_S32K1xx.h.

### 16.28.2.3 struct sim\_tclk\_config\_t

SIM Platform Gate Clock configuration. Implements sim\_tclk\_config\_t\_Class.

Definition at line 168 of file clock\_S32K1xx.h.

## Data Fields

- bool [initialize](#)
- uint32\_t [tclkFreq](#) [[NUMBER\\_OF\\_TCLK\\_INPUTS](#)]
- uint32\_t [extPinSrc](#) [[FTM\\_INSTANCE\\_COUNT](#)]

## Field Documentation

### 16.28.2.3.1 uint32\_t extPinSrc[FTM\_INSTANCE\_COUNT]

FTMx frequency.

Definition at line 172 of file clock\_S32K1xx.h.

### 16.28.2.3.2 bool initialize

Initialize or not the TCLK clock.

Definition at line 170 of file clock\_S32K1xx.h.

### 16.28.2.3.3 uint32\_t tclkFreq[NUMBER\_OF\_TCLK\_INPUTS]

TCLKx frequency.

Definition at line 171 of file clock\_S32K1xx.h.

#### 16.28.2.4 struct sim\_plat\_gate\_config\_t

SIM Platform Gate Clock configuration. Implements sim\_plat\_gate\_config\_t\_Class.

Definition at line 179 of file clock\_S32K1xx.h.

##### Data Fields

- bool [initialize](#)
- bool [enableMscm](#)
- bool [enableMpu](#)
- bool [enableDma](#)
- bool [enableErm](#)
- bool [enableEim](#)

##### Field Documentation

###### 16.28.2.4.1 bool enableDma

DMA Clock Gating Control enable.

Definition at line 184 of file clock\_S32K1xx.h.

###### 16.28.2.4.2 bool enableEim

EIM Clock Gating Control enable.

Definition at line 186 of file clock\_S32K1xx.h.

###### 16.28.2.4.3 bool enableErm

ERM Clock Gating Control enable.

Definition at line 185 of file clock\_S32K1xx.h.

###### 16.28.2.4.4 bool enableMpu

MPU Clock Gating Control enable.

Definition at line 183 of file clock\_S32K1xx.h.

###### 16.28.2.4.5 bool enableMscm

MSCM Clock Gating Control enable.

Definition at line 182 of file clock\_S32K1xx.h.

###### 16.28.2.4.6 bool initialize

Initialize or not the Trace clock.

Definition at line 181 of file clock\_S32K1xx.h.

#### 16.28.2.5 struct sim\_qspi\_ref\_clk\_gating\_t

SIM QSPI reference clock gating. Implements sim\_qspi\_ref\_clk\_gating\_t\_Class.

Definition at line 193 of file clock\_S32K1xx.h.

##### Data Fields

- bool [enableQspiRefClk](#)

##### Field Documentation

**16.28.2.5.1 bool enableQspiRefClk**

qspi internal reference clock gating control enable.

Definition at line 195 of file clock\_S32K1xx.h.

**16.28.2.6 struct sim\_trace\_clock\_config\_t**

SIM Debug Trace clock configuration. Implements sim\_trace\_clock\_config\_t\_Class.

Definition at line 213 of file clock\_S32K1xx.h.

**Data Fields**

- bool [initialize](#)
- bool [divEnable](#)
- [clock\\_trace\\_src\\_t](#) [source](#)
- [uint8\\_t](#) [divider](#)
- bool [divFraction](#)

**Field Documentation****16.28.2.6.1 bool divEnable**

Trace clock divider enable.

Definition at line 216 of file clock\_S32K1xx.h.

**16.28.2.6.2 bool divFraction**

Trace clock divider fraction.

Definition at line 219 of file clock\_S32K1xx.h.

**16.28.2.6.3 uint8\_t divider**

Trace clock divider divisor.

Definition at line 218 of file clock\_S32K1xx.h.

**16.28.2.6.4 bool initialize**

Initialize or not the Trace clock.

Definition at line 215 of file clock\_S32K1xx.h.

**16.28.2.6.5 clock\_trace\_src\_t source**

Trace clock select.

Definition at line 217 of file clock\_S32K1xx.h.

**16.28.2.7 struct sim\_clock\_config\_t**

SIM configure structure. Implements sim\_clock\_config\_t\_Class.

Definition at line 226 of file clock\_S32K1xx.h.

**Data Fields**

- [sim\\_clock\\_out\\_config\\_t](#) [clockOutConfig](#)
- [sim\\_lpo\\_clock\\_config\\_t](#) [lpoClockConfig](#)
- [sim\\_tclk\\_config\\_t](#) [tclkConfig](#)
- [sim\\_plat\\_gate\\_config\\_t](#) [platGateConfig](#)
- [sim\\_trace\\_clock\\_config\\_t](#) [traceClockConfig](#)
- [sim\\_qspi\\_ref\\_clk\\_gating\\_t](#) [qspiRefClkGating](#)

### Field Documentation

#### 16.28.2.7.1 `sim_clock_out_config_t` clockOutConfig

Clock Out configuration.

Definition at line 228 of file clock\_S32K1xx.h.

#### 16.28.2.7.2 `sim_lpo_clock_config_t` lpoClockConfig

Low Power Clock configuration.

Definition at line 229 of file clock\_S32K1xx.h.

#### 16.28.2.7.3 `sim_plat_gate_config_t` platGateConfig

Platform Gate Clock configuration.

Definition at line 231 of file clock\_S32K1xx.h.

#### 16.28.2.7.4 `sim_qspi_ref_clk_gating_t` qspiRefClkGating

Qspi Reference Clock Gating.

Definition at line 233 of file clock\_S32K1xx.h.

#### 16.28.2.7.5 `sim_tclk_config_t` tclkConfig

TCLK, FTM option Clock configuration.

Definition at line 230 of file clock\_S32K1xx.h.

#### 16.28.2.7.6 `sim_trace_clock_config_t` traceClockConfig

Trace clock configuration.

Definition at line 232 of file clock\_S32K1xx.h.

#### 16.28.2.8 `struct scg_system_clock_config_t`

SCG system clock configuration. Implements `scg_system_clock_config_t_Class`.

Definition at line 280 of file clock\_S32K1xx.h.

### Data Fields

- [scg\\_system\\_clock\\_div\\_t](#) divSlow
- [scg\\_system\\_clock\\_div\\_t](#) divBus
- [scg\\_system\\_clock\\_div\\_t](#) divCore
- [scg\\_system\\_clock\\_src\\_t](#) src

### Field Documentation

#### 16.28.2.8.1 `scg_system_clock_div_t` divBus

BUS clock divider.

Definition at line 283 of file clock\_S32K1xx.h.

#### 16.28.2.8.2 `scg_system_clock_div_t` divCore

Core clock divider.

Definition at line 284 of file clock\_S32K1xx.h.

#### 16.28.2.8.3 `scg_system_clock_div_t` divSlow

Slow clock divider.

Definition at line 282 of file `clock_S32K1xx.h`.

#### 16.28.2.8.4 `scg_system_clock_src_t` src

System clock source.

Definition at line 285 of file `clock_S32K1xx.h`.

#### 16.28.2.9 `struct scg_sosc_config_t`

SCG system OSC configuration. Implements `scg_sosc_config_t_Class`.

Definition at line 370 of file `clock_S32K1xx.h`.

##### Data Fields

- `uint32_t freq`
- `scg_sosc_monitor_mode_t monitorMode`
- `scg_sosc_ext_ref_t extRef`
- `scg_sosc_gain_t gain`
- `scg_sosc_range_t range`
- `scg_async_clock_div_t div1`
- `scg_async_clock_div_t div2`
- `bool enableInStop`
- `bool enableInLowPower`
- `bool locked`
- `bool initialize`

##### Field Documentation

#### 16.28.2.9.1 `scg_async_clock_div_t` div1

Asynchronous peripheral source.

Definition at line 381 of file `clock_S32K1xx.h`.

#### 16.28.2.9.2 `scg_async_clock_div_t` div2

Asynchronous peripheral source.

Definition at line 382 of file `clock_S32K1xx.h`.

#### 16.28.2.9.3 `bool enableInLowPower`

System OSC is enable or not in low power mode.

Definition at line 385 of file `clock_S32K1xx.h`.

#### 16.28.2.9.4 `bool enableInStop`

System OSC is enable or not in stop mode.

Definition at line 384 of file `clock_S32K1xx.h`.

#### 16.28.2.9.5 `scg_sosc_ext_ref_t` extRef

System OSC External Reference Select.

Definition at line 376 of file `clock_S32K1xx.h`.



**16.28.2.9.6 uint32\_t freq**

System OSC frequency.

Definition at line 372 of file clock\_S32K1xx.h.

**16.28.2.9.7 scg\_sosc\_gain\_t gain**

System OSC high-gain operation.

Definition at line 377 of file clock\_S32K1xx.h.

**16.28.2.9.8 bool initialize**

Initialize or not the System OSC module.

Definition at line 389 of file clock\_S32K1xx.h.

**16.28.2.9.9 bool locked**

System OSC Control Register can be written.

Definition at line 387 of file clock\_S32K1xx.h.

**16.28.2.9.10 scg\_sosc\_monitor\_mode\_t monitorMode**

System OSC Clock monitor mode.

Definition at line 374 of file clock\_S32K1xx.h.

**16.28.2.9.11 scg\_sosc\_range\_t range**

System OSC frequency range.

Definition at line 379 of file clock\_S32K1xx.h.

**16.28.2.10 struct scg\_sirc\_config\_t**

SCG slow IRC clock configuration. Implements scg\_sirc\_config\_t\_Class.

Definition at line 405 of file clock\_S32K1xx.h.

**Data Fields**

- [scg\\_sirc\\_range\\_t range](#)
- [scg\\_async\\_clock\\_div\\_t div1](#)
- [scg\\_async\\_clock\\_div\\_t div2](#)
- [bool initialize](#)
- [bool enableInStop](#)
- [bool enableInLowPower](#)
- [bool locked](#)

**Field Documentation****16.28.2.10.1 scg\_async\_clock\_div\_t div1**

Asynchronous peripheral source.

Definition at line 409 of file clock\_S32K1xx.h.

**16.28.2.10.2 scg\_async\_clock\_div\_t div2**

Asynchronous peripheral source.

Definition at line 410 of file clock\_S32K1xx.h.

**16.28.2.10.3 bool enableInLowPower**

SIRC is enable or not in low power mode.

Definition at line 414 of file clock\_S32K1xx.h.

**16.28.2.10.4 bool enableInStop**

SIRC is enable or not in stop mode.

Definition at line 413 of file clock\_S32K1xx.h.

**16.28.2.10.5 bool initialize**

Initialize or not the SIRC module.

Definition at line 412 of file clock\_S32K1xx.h.

**16.28.2.10.6 bool locked**

SIRC Control Register can be written.

Definition at line 416 of file clock\_S32K1xx.h.

**16.28.2.10.7 scg\_sirc\_range\_t range**

Slow IRC frequency range.

Definition at line 407 of file clock\_S32K1xx.h.

**16.28.2.11 struct scg\_firc\_config\_t**

SCG fast IRC clock configuration. Implements scg\_firc\_config\_t\_Class.

Definition at line 432 of file clock\_S32K1xx.h.

**Data Fields**

- [scg\\_firc\\_range\\_t range](#)
- [scg\\_async\\_clock\\_div\\_t div1](#)
- [scg\\_async\\_clock\\_div\\_t div2](#)
- bool [enableInStop](#)
- bool [enableInLowPower](#)
- bool [regulator](#)
- bool [locked](#)
- bool [initialize](#)

**Field Documentation****16.28.2.11.1 scg\_async\_clock\_div\_t div1**

Asynchronous peripheral source.

Definition at line 436 of file clock\_S32K1xx.h.

**16.28.2.11.2 scg\_async\_clock\_div\_t div2**

Asynchronous peripheral source.

Definition at line 437 of file clock\_S32K1xx.h.

**16.28.2.11.3 bool enableInLowPower**

FIRC is enable or not in lowpower mode.

Definition at line 440 of file clock\_S32K1xx.h.

**16.28.2.11.4 bool enableInStop**

FIRC is enable or not in stop mode.

Definition at line 439 of file clock\_S32K1xx.h.

**16.28.2.11.5 bool initialize**

Initialize or not the FIRC module.

Definition at line 444 of file clock\_S32K1xx.h.

**16.28.2.11.6 bool locked**

FIRC Control Register can be written.

Definition at line 442 of file clock\_S32K1xx.h.

**16.28.2.11.7 scg\_firc\_range\_t range**

Fast IRC frequency range.

Definition at line 434 of file clock\_S32K1xx.h.

**16.28.2.11.8 bool regulator**

FIRC regulator is enable or not.

Definition at line 441 of file clock\_S32K1xx.h.

**16.28.2.12 struct scg\_spll\_config\_t**

SCG system PLL configuration. Implements scg\_spll\_config\_t\_Class.

Definition at line 518 of file clock\_S32K1xx.h.

**Data Fields**

- [scg\\_spll\\_monitor\\_mode\\_t monitorMode](#)
- [uint8\\_t prediv](#)
- [uint8\\_t mult](#)
- [uint8\\_t src](#)
- [scg\\_async\\_clock\\_div\\_t div1](#)
- [scg\\_async\\_clock\\_div\\_t div2](#)
- [bool enableInStop](#)
- [bool locked](#)
- [bool initialize](#)

**Field Documentation****16.28.2.12.1 scg\_async\_clock\_div\_t div1**

Asynchronous peripheral source.

Definition at line 526 of file clock\_S32K1xx.h.

**16.28.2.12.2 scg\_async\_clock\_div\_t div2**

Asynchronous peripheral source.

Definition at line 527 of file clock\_S32K1xx.h.

**16.28.2.12.3 bool enableInStop**

System PLL clock is enable or not in stop mode.

Definition at line 529 of file clock\_S32K1xx.h.

#### 16.28.2.12.4 bool initialize

Initialize or not the System PLL module.

Definition at line 532 of file clock\_S32K1xx.h.

#### 16.28.2.12.5 bool locked

System PLL Control Register can be written.

Definition at line 531 of file clock\_S32K1xx.h.

#### 16.28.2.12.6 scg\_spill\_monitor\_mode\_t monitorMode

Clock monitor mode selected.

Definition at line 520 of file clock\_S32K1xx.h.

#### 16.28.2.12.7 uint8\_t mult

System PLL multiplier.

Definition at line 523 of file clock\_S32K1xx.h.

#### 16.28.2.12.8 uint8\_t prediv

PLL reference clock divider.

Definition at line 522 of file clock\_S32K1xx.h.

#### 16.28.2.12.9 uint8\_t src

System PLL source.

Definition at line 524 of file clock\_S32K1xx.h.

#### 16.28.2.13 struct scg\_rtc\_config\_t

SCG RTC configuration. Implements scg\_rtc\_config\_t\_Class.

Definition at line 539 of file clock\_S32K1xx.h.

##### Data Fields

- uint32\_t [rtcClkInFreq](#)
- bool [initialize](#)

##### Field Documentation

#### 16.28.2.13.1 bool initialize

Initialize or not the RTC.

Definition at line 542 of file clock\_S32K1xx.h.

#### 16.28.2.13.2 uint32\_t rtcClkInFreq

RTC\_CLKIN frequency.

Definition at line 541 of file clock\_S32K1xx.h.

#### 16.28.2.14 struct scg\_clock\_mode\_config\_t

SCG Clock Mode Configuration structure. Implements scg\_clock\_mode\_config\_t\_Class.

Definition at line 549 of file clock\_S32K1xx.h.

#### Data Fields

- [scg\\_system\\_clock\\_config\\_t rccrConfig](#)
- [scg\\_system\\_clock\\_config\\_t vccrConfig](#)
- [scg\\_system\\_clock\\_config\\_t hccrConfig](#)
- [scg\\_system\\_clock\\_src\\_t alternateClock](#)
- [bool initialize](#)

#### Field Documentation

##### 16.28.2.14.1 [scg\\_system\\_clock\\_src\\_t alternateClock](#)

Alternate clock used during initialization

Definition at line 554 of file clock\_S32K1xx.h.

##### 16.28.2.14.2 [scg\\_system\\_clock\\_config\\_t hccrConfig](#)

HSRUN Clock Control configuration.

Definition at line 553 of file clock\_S32K1xx.h.

##### 16.28.2.14.3 [bool initialize](#)

Initialize or not the Clock Mode Configuration.

Definition at line 555 of file clock\_S32K1xx.h.

##### 16.28.2.14.4 [scg\\_system\\_clock\\_config\\_t rccrConfig](#)

Run Clock Control configuration.

Definition at line 551 of file clock\_S32K1xx.h.

##### 16.28.2.14.5 [scg\\_system\\_clock\\_config\\_t vccrConfig](#)

VLPR Clock Control configuration.

Definition at line 552 of file clock\_S32K1xx.h.

##### 16.28.2.15 [struct scg\\_clockout\\_config\\_t](#)

SCG ClockOut Configuration structure. Implements [scg\\_clockout\\_config\\_t\\_Class](#).

Definition at line 562 of file clock\_S32K1xx.h.

#### Data Fields

- [scg\\_clockout\\_src\\_t source](#)
- [bool initialize](#)

#### Field Documentation

##### 16.28.2.15.1 [bool initialize](#)

Initialize or not the ClockOut.

Definition at line 565 of file clock\_S32K1xx.h.

##### 16.28.2.15.2 [scg\\_clockout\\_src\\_t source](#)

ClockOut source select.

Definition at line 564 of file clock\_S32K1xx.h.

### 16.28.2.16 struct scg\_config\_t

SCG configure structure. Implements scg\_config\_t\_Class.

Definition at line 572 of file clock\_S32K1xx.h.

#### Data Fields

- [scg\\_sirc\\_config\\_t sircConfig](#)
- [scg\\_firc\\_config\\_t fircConfig](#)
- [scg\\_sosc\\_config\\_t soscConfig](#)
- [scg\\_spill\\_config\\_t spillConfig](#)
- [scg\\_rtc\\_config\\_t rtcConfig](#)
- [scg\\_clockout\\_config\\_t clockOutConfig](#)
- [scg\\_clock\\_mode\\_config\\_t clockModeConfig](#)

#### Field Documentation

##### 16.28.2.16.1 scg\_clock\_mode\_config\_t clockModeConfig

SCG Clock Mode Configuration.

Definition at line 580 of file clock\_S32K1xx.h.

##### 16.28.2.16.2 scg\_clockout\_config\_t clockOutConfig

SCG ClockOut Configuration.

Definition at line 579 of file clock\_S32K1xx.h.

##### 16.28.2.16.3 scg\_firc\_config\_t fircConfig

Fast internal reference clock configuration.

Definition at line 575 of file clock\_S32K1xx.h.

##### 16.28.2.16.4 scg\_rtc\_config\_t rtcConfig

Real Time Clock configuration.

Definition at line 578 of file clock\_S32K1xx.h.

##### 16.28.2.16.5 scg\_sirc\_config\_t sircConfig

Slow internal reference clock configuration.

Definition at line 574 of file clock\_S32K1xx.h.

##### 16.28.2.16.6 scg\_sosc\_config\_t soscConfig

System oscillator configuration.

Definition at line 576 of file clock\_S32K1xx.h.

##### 16.28.2.16.7 scg\_spill\_config\_t spillConfig

System Phase locked loop configuration.

Definition at line 577 of file clock\_S32K1xx.h.

### 16.28.2.17 struct peripheral\_clock\_config\_t

PCC peripheral instance clock configuration. Implements peripheral\_clock\_config\_t\_Class.

Definition at line 632 of file clock\_S32K1xx.h.

#### Data Fields

- `clock_names_t` [clockName](#)
- `bool` [clkGate](#)
- `peripheral_clock_source_t` [clkSrc](#)
- `peripheral_clock_frac_t` [frac](#)
- `peripheral_clock_divider_t` [divider](#)

#### Field Documentation

##### 16.28.2.17.1 `bool` [clkGate](#)

Peripheral clock gate.

Definition at line 642 of file `clock_S32K1xx.h`.

##### 16.28.2.17.2 `peripheral_clock_source_t` [clkSrc](#)

Peripheral clock source.

Definition at line 643 of file `clock_S32K1xx.h`.

##### 16.28.2.17.3 `clock_names_t` [clockName](#)

Definition at line 641 of file `clock_S32K1xx.h`.

##### 16.28.2.17.4 `peripheral_clock_divider_t` [divider](#)

Peripheral clock divider value.

Definition at line 645 of file `clock_S32K1xx.h`.

##### 16.28.2.17.5 `peripheral_clock_frac_t` [frac](#)

Peripheral clock fractional value.

Definition at line 644 of file `clock_S32K1xx.h`.

##### 16.28.2.18 `struct` [pcc\\_config\\_t](#)

PCC configuration. Implements `pcc_config_t_Class`.

Definition at line 651 of file `clock_S32K1xx.h`.

#### Data Fields

- `uint32_t` [count](#)
- `peripheral_clock_config_t *` [peripheralClocks](#)

#### Field Documentation

##### 16.28.2.18.1 `uint32_t` [count](#)

Number of peripherals to be configured.

Definition at line 653 of file `clock_S32K1xx.h`.

##### 16.28.2.18.2 `peripheral_clock_config_t*` [peripheralClocks](#)

Pointer to the peripheral clock configurations array.

Definition at line 654 of file `clock_S32K1xx.h`.

### 16.28.2.19 struct pmc\_lpo\_clock\_config\_t

PMC LPO configuration. Implements pmc\_lpo\_clock\_config\_t\_Class.

Definition at line 660 of file clock\_S32K1xx.h.

#### Data Fields

- bool [initialize](#)
- bool [enable](#)
- int8\_t [trimValue](#)

#### Field Documentation

##### 16.28.2.19.1 bool enable

Enable/disable LPO

Definition at line 663 of file clock\_S32K1xx.h.

##### 16.28.2.19.2 bool initialize

Initialize or not the PMC LPO settings.

Definition at line 662 of file clock\_S32K1xx.h.

##### 16.28.2.19.3 int8\_t trimValue

LPO trimming value

Definition at line 664 of file clock\_S32K1xx.h.

### 16.28.2.20 struct pmc\_config\_t

PMC configure structure. Implements pmc\_config\_t\_Class.

Definition at line 671 of file clock\_S32K1xx.h.

#### Data Fields

- [pmc\\_lpo\\_clock\\_config\\_t](#) lpoClockConfig

#### Field Documentation

##### 16.28.2.20.1 pmc\_lpo\_clock\_config\_t lpoClockConfig

Low Power Clock configuration.

Definition at line 673 of file clock\_S32K1xx.h.

##### 16.28.2.21 struct clock\_manager\_user\_config\_t

Clock configuration structure. Implements clock\_manager\_user\_config\_t\_Class.

Definition at line 680 of file clock\_S32K1xx.h.

#### Data Fields

- [scg\\_config\\_t](#) scgConfig
- [sim\\_clock\\_config\\_t](#) simConfig
- [pcc\\_config\\_t](#) pccConfig
- [pmc\\_config\\_t](#) pmcConfig



### Field Documentation

#### 16.28.2.21.1 pcc\_config\_t pccConfig

PCC Clock configuration.

Definition at line 684 of file clock\_S32K1xx.h.

#### 16.28.2.21.2 pmc\_config\_t pmcConfig

PMC Clock configuration.

Definition at line 685 of file clock\_S32K1xx.h.

#### 16.28.2.21.3 scg\_config\_t scgConfig

SCG Clock configuration.

Definition at line 682 of file clock\_S32K1xx.h.

#### 16.28.2.21.4 sim\_clock\_config\_t simConfig

SIM Clock configuration.

Definition at line 683 of file clock\_S32K1xx.h.

#### 16.28.2.22 struct module\_clk\_config\_t

module clock configuration. Implements module\_clk\_config\_t\_Class

Definition at line 720 of file clock\_S32K1xx.h.

### Data Fields

- bool [gating](#)
- clock\_names\_t [source](#)
- uint16\_t [mul](#)
- uint16\_t [div](#)

### Field Documentation

#### 16.28.2.22.1 uint16\_t div

Divider (some modules don't have divider)

Definition at line 725 of file clock\_S32K1xx.h.

#### 16.28.2.22.2 bool gating

Clock gating.

Definition at line 722 of file clock\_S32K1xx.h.

#### 16.28.2.22.3 uint16\_t mul

Multiplier (some modules don't have fractional)

Definition at line 724 of file clock\_S32K1xx.h.

#### 16.28.2.22.4 clock\_names\_t source

Clock source input (some modules don't have protocol clock)

Definition at line 723 of file clock\_S32K1xx.h.

### 16.28.2.23 struct sys\_clk\_config\_t

System clock configuration. Implements sys\_clk\_config\_t\_Class.

Definition at line 733 of file clock\_S32K1xx.h.

#### Data Fields

- clock\_names\_t [src](#)
- uint16\_t [dividers](#) [[SYS\\_CLK\\_MAX\\_NO](#)]

#### Field Documentation

#### 16.28.2.23.1 uint16\_t dividers[SYS\_CLK\_MAX\_NO]

System clock dividers. Value by which system clock is divided. 0 means that system clock is not divided.

Definition at line 736 of file clock\_S32K1xx.h.

#### 16.28.2.23.2 clock\_names\_t src

System clock source.

Definition at line 735 of file clock\_S32K1xx.h.

### 16.28.2.24 struct clock\_source\_config\_t

Clock source configuration. Implements clock\_source\_config\_t\_Class.

Definition at line 743 of file clock\_S32K1xx.h.

#### Data Fields

- bool [enable](#)
- [xosc\\_ref\\_t](#) [refClk](#)
- [uint32\\_t](#) [refFreq](#)
- [uint16\\_t](#) [mul](#)
- [uint16\\_t](#) [div](#)
- [uint16\\_t](#) [outputDiv1](#)
- [uint16\\_t](#) [outputDiv2](#)

#### Field Documentation

#### 16.28.2.24.1 uint16\_t div

Divider. It applies to PLL clock sources. Valid range is 1-8.

Definition at line 749 of file clock\_S32K1xx.h.

#### 16.28.2.24.2 bool enable

Enable/disable clock source.

Definition at line 745 of file clock\_S32K1xx.h.

#### 16.28.2.24.3 uint16\_t mul

Multiplier. It applies to PLL clock sources. Valid range is 16 - 47.

Definition at line 748 of file clock\_S32K1xx.h.

#### 16.28.2.24.4 uint16\_t outputDiv1

First output divider. It's used as protocol clock by modules. Zero means that divider is disabled. / Possible values 0(disabled), 1, 2, 4, 8, 16, 32, 64; all the other values are not valid. /

Definition at line 751 of file clock\_S32K1xx.h.

#### 16.28.2.24.5 `uint16_t` outputDiv2

Second output divider. It's used as protocol clock by modules. Zero means that divider is disabled. / Possible values 0(disabled), 1, 2, 4, 8, 16, 32, 64; all the other values are not valid. /

Definition at line 754 of file clock\_S32K1xx.h.

#### 16.28.2.24.6 `xosc_ref_t` refClk

Bypass option. It applies to external oscillator clock sources

Definition at line 746 of file clock\_S32K1xx.h.

#### 16.28.2.24.7 `uint32_t` refFreq

Frequency of the input reference clock. It applies to external oscillator clock sources

Definition at line 747 of file clock\_S32K1xx.h.

#### 16.28.2.25 `struct clock_notify_struct_t`

Clock notification structure passed to clock callback function. Implements `clock_notify_struct_t_Class`.

Definition at line 797 of file clock\_S32K1xx.h.

##### Data Fields

- `uint8_t` [targetClockConfigIndex](#)
- `clock_manager_policy_t` [policy](#)
- `clock_manager_notify_t` [notifyType](#)

##### Field Documentation

#### 16.28.2.25.1 `clock_manager_notify_t` notifyType

Clock notification type.

Definition at line 801 of file clock\_S32K1xx.h.

#### 16.28.2.25.2 `clock_manager_policy_t` policy

Clock transition policy.

Definition at line 800 of file clock\_S32K1xx.h.

#### 16.28.2.25.3 `uint8_t` targetClockConfigIndex

Target clock configuration index.

Definition at line 799 of file clock\_S32K1xx.h.

#### 16.28.2.26 `struct clock_manager_callback_user_config_t`

Structure for callback function and its parameter. Implements `clock_manager_callback_user_config_t_Class`.

Definition at line 814 of file clock\_S32K1xx.h.

##### Data Fields

- `clock_manager_callback_t` [callback](#)
- `clock_manager_callback_type_t` [callbackType](#)
- `void *` [callbackData](#)

## Field Documentation

### 16.28.2.26.1 `clock_manager_callback_t` callback

Entry of callback function.

Definition at line 816 of file `clock_S32K1xx.h`.

### 16.28.2.26.2 `void* callbackData`

Parameter of callback function.

Definition at line 818 of file `clock_S32K1xx.h`.

### 16.28.2.26.3 `clock_manager_callback_type_t` callbackType

Callback type.

Definition at line 817 of file `clock_S32K1xx.h`.

### 16.28.2.27 `struct clock_manager_state_t`

Clock manager state structure. Implements `clock_manager_state_t_Class`.

Definition at line 825 of file `clock_S32K1xx.h`.

## Data Fields

- `clock_manager_user_config_t` const \*\* `configTable`
- `uint8_t` `clockConfigNum`
- `uint8_t` `curConfigIndex`
- `clock_manager_callback_user_config_t` \*\* `callbackConfig`
- `uint8_t` `callbackNum`
- `uint8_t` `errorCallbackIndex`

## Field Documentation

### 16.28.2.27.1 `clock_manager_callback_user_config_t**` callbackConfig

Pointer to callback table.

Definition at line 830 of file `clock_S32K1xx.h`.

### 16.28.2.27.2 `uint8_t` callbackNum

Number of clock callbacks.

Definition at line 831 of file `clock_S32K1xx.h`.

### 16.28.2.27.3 `uint8_t` clockConfigNum

Number of clock configurations.

Definition at line 828 of file `clock_S32K1xx.h`.

### 16.28.2.27.4 `clock_manager_user_config_t` const\*\* configTable

Pointer to clock configure table.

Definition at line 827 of file `clock_S32K1xx.h`.

### 16.28.2.27.5 `uint8_t` curConfigIndex

Index of current configuration.

Definition at line 829 of file `clock_S32K1xx.h`.

**16.28.2.27.6 uint8\_t errorCallbackIndex**

Index of callback returns error.

Definition at line 832 of file clock\_S32K1xx.h.

**16.28.3 Macro Definition Documentation****16.28.3.1 #define BUS\_CLK\_INDEX 1U**

Definition at line 69 of file clock\_S32K1xx.h.

**16.28.3.2 #define CLK\_SRC\_FIRC 0x03U**

SCGFIRCLK - Fast IRC Clock

Definition at line 591 of file clock\_S32K1xx.h.

**16.28.3.3 #define CLK\_SRC\_FIRC\_DIV1 0x03U**

SCGFIRCLK - Fast IRC Clock

Definition at line 595 of file clock\_S32K1xx.h.

**16.28.3.4 #define CLK\_SRC\_FIRC\_DIV2 0x03U**

SCGFIRCLK - Fast IRC Clock

Definition at line 599 of file clock\_S32K1xx.h.

**16.28.3.5 #define CLK\_SRC\_OFF 0x00U**

Clock is off

Definition at line 588 of file clock\_S32K1xx.h.

**16.28.3.6 #define CLK\_SRC\_SIRC 0x02U**

SCGIRCLK - Slow IRC Clock

Definition at line 590 of file clock\_S32K1xx.h.

**16.28.3.7 #define CLK\_SRC\_SIRC\_DIV1 0x02U**

SCGIRCLK - Slow IRC Clock

Definition at line 594 of file clock\_S32K1xx.h.

**16.28.3.8 #define CLK\_SRC\_SIRC\_DIV2 0x02U**

SCGIRCLK - Slow IRC Clock

Definition at line 598 of file clock\_S32K1xx.h.

**16.28.3.9 #define CLK\_SRC\_SOSC 0x01U**

OSCCLK - System Oscillator Bus Clock

Definition at line 589 of file clock\_S32K1xx.h.

**16.28.3.10 #define CLK\_SRC\_SOSC\_DIV1 0x01U**

OSCCLK - System Oscillator Bus Clock

Definition at line 593 of file clock\_S32K1xx.h.

**16.28.3.11 #define CLK\_SRC\_SOSC\_DIV2 0x01U**

OSCCLK - System Oscillator Bus Clock

Definition at line 597 of file clock\_S32K1xx.h.

**16.28.3.12 #define CLK\_SRC\_SPLL 0x06U**

SCGPCLK System PLL clock

Definition at line 592 of file clock\_S32K1xx.h.

**16.28.3.13 #define CLK\_SRC\_SPLL\_DIV1 0x06U**

SCGPCLK System PLL clock

Definition at line 596 of file clock\_S32K1xx.h.

**16.28.3.14 #define CLK\_SRC\_SPLL\_DIV2 0x06U**

SCGPCLK System PLL clock

Definition at line 600 of file clock\_S32K1xx.h.

**16.28.3.15 #define CORE\_CLK\_INDEX 0U**

Definition at line 68 of file clock\_S32K1xx.h.

**16.28.3.16 #define NUMBER\_OF\_TCLK\_INPUTS 3U**

TCLK clock frequency.

Definition at line 57 of file clock\_S32K1xx.h.

**16.28.3.17 #define SLOW\_CLK\_INDEX 2U**

Definition at line 70 of file clock\_S32K1xx.h.

**16.28.3.18 #define SYS\_CLK\_MAX\_NO 3U**

The maximum number of system clock dividers and system clock divider indexes.

Definition at line 67 of file clock\_S32K1xx.h.

**16.28.4 Typedef Documentation****16.28.4.1 typedef status\_t(\* clock\_manager\_callback\_t) (clock\_notify\_struct\_t \*notify, void \*callbackData)**

Type of clock callback functions.

Definition at line 807 of file clock\_S32K1xx.h.

**16.28.4.2 typedef clock\_manager\_user\_config\_t clock\_user\_config\_t**

Definition at line 688 of file clock\_S32K1xx.h.

**16.28.4.3 typedef uint8\_t peripheral\_clock\_source\_t**

PCC clock source select Implements peripheral\_clock\_source\_t\_Class.

Definition at line 586 of file clock\_S32K1xx.h.

**16.28.5 Enumeration Type Documentation**

## 16.28.5.1 enum clock\_manager\_callback\_type\_t

The callback type, indicates what kinds of notification this callback handles. Implements clock\_manager\_callback\_type\_t\_Class.

## Enumerator

**CLOCK\_MANAGER\_CALLBACK\_BEFORE** Callback handles BEFORE notification.

**CLOCK\_MANAGER\_CALLBACK\_AFTER** Callback handles AFTER notification.

**CLOCK\_MANAGER\_CALLBACK\_BEFORE\_AFTER** Callback handles BEFORE and AFTER notification

Definition at line 776 of file clock\_S32K1xx.h.

## 16.28.5.2 enum clock\_manager\_notify\_t

The clock notification type. Implements clock\_manager\_notify\_t\_Class.

## Enumerator

**CLOCK\_MANAGER\_NOTIFY\_RECOVER** Notify IP to recover to previous work state.

**CLOCK\_MANAGER\_NOTIFY\_BEFORE** Notify IP that system will change clock setting.

**CLOCK\_MANAGER\_NOTIFY\_AFTER** Notify IP that have changed to new clock setting.

Definition at line 765 of file clock\_S32K1xx.h.

## 16.28.5.3 enum clock\_manager\_policy\_t

Clock transition policy. Implements clock\_manager\_policy\_t\_Class.

## Enumerator

**CLOCK\_MANAGER\_POLICY\_AGREEMENT** Clock transfers gracefully.

**CLOCK\_MANAGER\_POLICY\_FORCIBLE** Clock transfers forcefully.

Definition at line 787 of file clock\_S32K1xx.h.

## 16.28.5.4 enum clock\_trace\_src\_t

Debug trace clock source select Implements clock\_trace\_src\_t\_Class.

## Enumerator

**CLOCK\_TRACE\_SRC\_CORE\_CLK** core clock

Definition at line 203 of file clock\_S32K1xx.h.

## 16.28.5.5 enum peripheral\_clock\_divider\_t

PCC divider value select Implements peripheral\_clock\_divider\_t\_Class.

## Enumerator

**DIVIDE\_BY\_ONE** Divide by 1 (pass-through, no clock divide)

**DIVIDE\_BY\_TWO** Divide by 2

**DIVIDE\_BY\_THREE** Divide by 3

**DIVIDE\_BY\_FOUR** Divide by 4

**DIVIDE\_BY\_FIVE** Divide by 5

**DIVIDE\_BY\_SIX** Divide by 6

**DIVIDE\_BY\_SEVEN** Divide by 7

**DIVIDE\_BY\_EIGHT** Divide by 8

Definition at line 617 of file clock\_S32K1xx.h.

## 16.28.5.6 enum peripheral\_clock\_frac\_t

PCC fractional value select Implements peripheral\_clock\_frac\_t\_Class.

## Enumerator

**MULTIPLY\_BY\_ONE** Fractional value is zero

**MULTIPLY\_BY\_TWO** Fractional value is one

Definition at line 608 of file clock\_S32K1xx.h.

## 16.28.5.7 enum pwr\_modes\_t

Power mode. Implements pwr\_modes\_t\_Class.

## Enumerator

**NO\_MODE**

**RUN\_MODE**

**VLPR\_MODE**

**HSRUN\_MODE**

**STOP\_MODE**

**VLPS\_MODE**

**ALL\_MODES**

Definition at line 694 of file clock\_S32K1xx.h.

## 16.28.5.8 enum scg\_async\_clock\_div\_t

SCG asynchronous clock divider value. Implements scg\_async\_clock\_div\_t\_Class.

## Enumerator

**SCG\_ASYNC\_CLOCK\_DISABLE** Clock output is disabled.

**SCG\_ASYNC\_CLOCK\_DIV\_BY\_1** Divided by 1.

**SCG\_ASYNC\_CLOCK\_DIV\_BY\_2** Divided by 2.

**SCG\_ASYNC\_CLOCK\_DIV\_BY\_4** Divided by 4.

**SCG\_ASYNC\_CLOCK\_DIV\_BY\_8** Divided by 8.

**SCG\_ASYNC\_CLOCK\_DIV\_BY\_16** Divided by 16.

**SCG\_ASYNC\_CLOCK\_DIV\_BY\_32** Divided by 32.

**SCG\_ASYNC\_CLOCK\_DIV\_BY\_64** Divided by 64.

Definition at line 312 of file clock\_S32K1xx.h.

## 16.28.5.9 enum scg\_clockout\_src\_t

SCG ClockOut type. Implements scg\_clockout\_src\_t\_Class.

## Enumerator

**SCG\_CLOCKOUT\_SRC\_SCG\_SLOW** SCG SLOW.

**SCG\_CLOCKOUT\_SRC\_SOSC** System OSC.

**SCG\_CLOCKOUT\_SRC\_SIRC** Slow IRC.

**SCG\_CLOCKOUT\_SRC\_FIRC** Fast IRC.

**SCG\_CLOCKOUT\_SRC\_SPLL** System PLL.

Definition at line 297 of file clock\_S32K1xx.h.



16.28.5.10 enum `scg_firc_range_t`

SCG fast IRC clock frequency range. Implements `scg_firc_range_t_Class`.

Enumerator

***SCG\_FIRC\_RANGE\_48M*** Fast IRC is trimmed to 48MHz.

Definition at line 423 of file `clock_S32K1xx.h`.

16.28.5.11 enum `scg_sirc_range_t`

SCG slow IRC clock frequency range. Implements `scg_sirc_range_t_Class`.

Enumerator

***SCG\_SIRC\_RANGE\_HIGH*** Slow IRC high range clock (8 MHz).

Definition at line 396 of file `clock_S32K1xx.h`.

16.28.5.12 enum `scg_sosc_ext_ref_t`

SCG OSC external reference clock select. Implements `scg_sosc_ext_ref_t_Class`.

Enumerator

***SCG\_SOSC\_REF\_EXT*** External reference clock requested

***SCG\_SOSC\_REF\_OSC*** Internal oscillator of OSC requested.

Definition at line 360 of file `clock_S32K1xx.h`.

16.28.5.13 enum `scg_sosc_gain_t`

SCG OSC high gain oscillator select. Implements `scg_sosc_gain_t_Class`.

Enumerator

***SCG\_SOSC\_GAIN\_LOW*** Configure crystal oscillator for low-power operation

***SCG\_SOSC\_GAIN\_HIGH*** Configure crystal oscillator for high-gain operation

Definition at line 350 of file `clock_S32K1xx.h`.

16.28.5.14 enum `scg_sosc_monitor_mode_t`

SCG system OSC monitor mode. Implements `scg_sosc_monitor_mode_t_Class`.

Enumerator

***SCG\_SOSC\_MONITOR\_DISABLE*** Monitor disable.

***SCG\_SOSC\_MONITOR\_INT*** Interrupt when system OSC error detected.

***SCG\_SOSC\_MONITOR\_RESET*** Reset when system OSC error detected.

Definition at line 329 of file `clock_S32K1xx.h`.

16.28.5.15 enum `scg_sosc_range_t`

SCG OSC frequency range select Implements `scg_sosc_range_t_Class`.

Enumerator

***SCG\_SOSC\_RANGE\_MID*** Medium frequency range selected for the crystal OSC (4 Mhz to 8 Mhz).

***SCG\_SOSC\_RANGE\_HIGH*** High frequency range selected for the crystal OSC (8 Mhz to 40 Mhz).

Definition at line 340 of file `clock_S32K1xx.h`.

## 16.28.5.16 enum scg\_spll\_clock\_multiply\_t

SCG system PLL multiplier.

## Enumerator

**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_16**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_17**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_18**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_19**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_20**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_21**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_22**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_23**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_24**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_25**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_26**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_27**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_28**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_29**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_30**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_31**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_32**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_33**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_34**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_35**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_36**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_37**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_38**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_39**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_40**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_41**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_42**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_43**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_44**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_45**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_46**  
**SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_47**

Definition at line 478 of file clock\_S32K1xx.h.

## 16.28.5.17 enum scg\_spll\_clock\_prediv\_t

SCG system PLL predivider.

## Enumerator

**SCG\_SPLL\_CLOCK\_PREDIV\_BY\_1**  
**SCG\_SPLL\_CLOCK\_PREDIV\_BY\_2**

**SCG\_SPLL\_CLOCK\_PREDIV\_BY\_3**  
**SCG\_SPLL\_CLOCK\_PREDIV\_BY\_4**  
**SCG\_SPLL\_CLOCK\_PREDIV\_BY\_5**  
**SCG\_SPLL\_CLOCK\_PREDIV\_BY\_6**  
**SCG\_SPLL\_CLOCK\_PREDIV\_BY\_7**  
**SCG\_SPLL\_CLOCK\_PREDIV\_BY\_8**

Definition at line 462 of file clock\_S32K1xx.h.

#### 16.28.5.18 enum scg\_spll\_monitor\_mode\_t

SCG system PLL monitor mode. Implements scg\_spll\_monitor\_mode\_t\_Class.

##### Enumerator

**SCG\_SPLL\_MONITOR\_DISABLE** Monitor disable.  
**SCG\_SPLL\_MONITOR\_INT** Interrupt when system PLL error detected.  
**SCG\_SPLL\_MONITOR\_RESET** Reset when system PLL error detected.

Definition at line 451 of file clock\_S32K1xx.h.

#### 16.28.5.19 enum scg\_system\_clock\_div\_t

SCG system clock divider value. Implements scg\_system\_clock\_div\_t\_Class.

##### Enumerator

**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_1** Divided by 1.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_2** Divided by 2.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_3** Divided by 3.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_4** Divided by 4.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_5** Divided by 5.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_6** Divided by 6.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_7** Divided by 7.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_8** Divided by 8.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_9** Divided by 9.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_10** Divided by 10.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_11** Divided by 11.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_12** Divided by 12.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_13** Divided by 13.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_14** Divided by 14.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_15** Divided by 15.  
**SCG\_SYSTEM\_CLOCK\_DIV\_BY\_16** Divided by 16.

Definition at line 256 of file clock\_S32K1xx.h.

#### 16.28.5.20 enum scg\_system\_clock\_src\_t

SCG system clock source. Implements scg\_system\_clock\_src\_t\_Class.

##### Enumerator

**SCG\_SYSTEM\_CLOCK\_SRC\_SYS\_OSC** System OSC.  
**SCG\_SYSTEM\_CLOCK\_SRC\_SIRC** Slow IRC.  
**SCG\_SYSTEM\_CLOCK\_SRC\_FIRC** Fast IRC.  
**SCG\_SYSTEM\_CLOCK\_SRC\_NONE** MAX value.

Definition at line 241 of file clock\_S32K1xx.h.

#### 16.28.5.21 enum `sim_clkout_div_t`

SIM CLKOUT divider Implements `sim_clkout_div_t_Class`.

##### Enumerator

**`SIM_CLKOUT_DIV_BY_1`** Divided by 1  
**`SIM_CLKOUT_DIV_BY_2`** Divided by 2  
**`SIM_CLKOUT_DIV_BY_3`** Divided by 3  
**`SIM_CLKOUT_DIV_BY_4`** Divided by 4  
**`SIM_CLKOUT_DIV_BY_5`** Divided by 5  
**`SIM_CLKOUT_DIV_BY_6`** Divided by 6  
**`SIM_CLKOUT_DIV_BY_7`** Divided by 7  
**`SIM_CLKOUT_DIV_BY_8`** Divided by 8

Definition at line 125 of file `clock_S32K1xx.h`.

#### 16.28.5.22 enum `sim_clkout_src_t`

SIM CLKOUT select Implements `sim_clkout_src_t_Class`.

##### Enumerator

**`SIM_CLKOUT_SEL_SYSTEM_SCG_CLKOUT`** SCG CLKOUT  
**`SIM_CLKOUT_SEL_SYSTEM_SOSC_DIV2_CLK`** SOSC DIV2 CLK  
**`SIM_CLKOUT_SEL_SYSTEM_SIRC_DIV2_CLK`** SIRC DIV2 CLK  
**`SIM_CLKOUT_SEL_SYSTEM_FIRC_DIV2_CLK`** FIRC DIV2 CLK  
**`SIM_CLKOUT_SEL_SYSTEM_HCLK`** HCLK  
**`SIM_CLKOUT_SEL_SYSTEM_SPLL_DIV2_CLK`** SPLL DIV2 CLK  
**`SIM_CLKOUT_SEL_SYSTEM_BUS_CLK`** BUS\_CLK  
**`SIM_CLKOUT_SEL_SYSTEM_LPO_128K_CLK`** LPO\_CLK 128 Khz  
**`SIM_CLKOUT_SEL_SYSTEM_LPO_CLK`** LPO\_CLK as selected by SIM LPO CLK Select  
**`SIM_CLKOUT_SEL_SYSTEM_RTC_CLK`** RTC CLK as selected by SIM CLK 32 KHz Select

Definition at line 100 of file `clock_S32K1xx.h`.

#### 16.28.5.23 enum `sim_lpoclk_sel_src_t`

SIM LPOCLKSEL clock source select Implements `sim_lpoclk_sel_src_t_Class`.

##### Enumerator

**`SIM_LPO_CLK_SEL_LPO_128K`** 128 kHz LPO clock  
**`SIM_LPO_CLK_SEL_NO_CLOCK`** No clock  
**`SIM_LPO_CLK_SEL_LPO_32K`** 32 kHz LPO clock which is divided by the 128 kHz LPO clock  
**`SIM_LPO_CLK_SEL_LPO_1K`** 1 kHz LPO clock which is divided by the 128 kHz LPO clock

Definition at line 88 of file `clock_S32K1xx.h`.

16.28.5.24 enum `sim_rtc_clk_sel_src_t`

SIM CLK32KSEL clock source select Implements `sim_rtc_clk_sel_src_t_Class`.

## Enumerator

***SIM\_RTCCLK\_SEL\_SOSCDIV1\_CLK*** SOSCDIV1 clock

***SIM\_RTCCLK\_SEL\_LPO\_32K*** 32 kHz LPO clock

***SIM\_RTCCLK\_SEL\_RTC\_CLKIN*** RTC\_CLKIN clock

***SIM\_RTCCLK\_SEL\_FIRCDIV1\_CLK*** FIRCDIV1 clock

Definition at line 76 of file `clock_S32K1xx.h`.

16.28.5.25 enum `xosc_ref_t`

XOSC reference clock select (internal oscillator is bypassed or not) Implements `xosc_ref_t_Class`.

## Enumerator

***XOSC\_EXT\_REF*** Internal oscillator is bypassed, external reference clock requested.

***XOSC\_INT\_OSC*** Internal oscillator of XOSC requested.

Definition at line 711 of file `clock_S32K1xx.h`.

## 16.28.6 Function Documentation

16.28.6.1 void `CLOCK_DRV_GetSystemClockSource ( sys_clk_config_t * sysClkConfig )`

Gets the system clock source.

This function gets the current system clock source.

## Returns

Value of the current system clock source.

Definition at line 3713 of file `clock_S32K1xx.c`.

16.28.6.2 status\_t `CLOCK_DRV_SetClockSource ( clock_names_t clockSource, const clock_source_config_t * clkSrcConfig )`

This function configures a clock source.

The clock source is configured based on the provided configuration. All values from the previous configuration of clock source are overwritten. If no configuration is provided, then a default one is used.

## Parameters

in	<i>clockSource</i>	Clock name of the configured clock source
in	<i>clkSrcConfig</i>	Pointer to the configuration structure

## Returns

Status of clock source initialization

Definition at line 4065 of file `clock_S32K1xx.c`.

16.28.6.3 void CLOCK\_DRV\_SetModuleClock ( clock\_names\_t *peripheralClock*, const module\_clk\_config\_t \*  
*moduleClkConfig* )

Configures module clock.

This function configures a module clock according to the configuration. If no configuration is provided (*moduleClkConfig* is null), then a default one is used *moduleClkConfig* must be passed as null when module doesn't support protocol clock.

**Parameters**

in	<i>peripheralClock</i>	Clock name of the configured module clock
in	<i>moduleClkConfig</i>	Pointer to the configuration structure.

Definition at line 3490 of file clock\_S32K1xx.c.

**16.28.6.4** `status_t CLOCK_DRV_SetSystemClock ( const pwr_modes_t * mode, const sys_clk_config_t * sysClkConfig )`

Configures the system clocks.

This function configures the system clocks (core, bus and flash clocks) in the specified power mode. If no power mode is specified (null parameter) then it is the current power mode.

**Parameters**

in	<i>mode</i>	Pointer to power mode for which the configured system clocks apply
in	<i>sysClkConfig</i>	Pointer to the system clocks configuration structure.

Definition at line 3644 of file clock\_S32K1xx.c.

**16.28.6.5** `uint8_t CLOCK_SYS_GetCurrentConfiguration ( void )`

Get current system clock configuration.

**Returns**

Current clock configuration index.

Definition at line 4292 of file clock\_S32K1xx.c.

**16.28.6.6** `clock_manager_callback_user_config_t* CLOCK_SYS_GetErrorCallback ( void )`

Get the callback which returns error in last clock switch.

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.

Definition at line 4304 of file clock\_S32K1xx.c.

**16.28.6.7** `status_t CLOCK_SYS_GetFreq ( clock_names_t clockName, uint32_t * frequency )`

Wrapper over CLOCK\_DRV\_GetFreq function. It's part of the old API.

**Parameters**

in	<i>clockName</i>	Clock names defined in clock_names_t
out	<i>frequency</i>	Returned clock frequency value in Hertz

**Returns**

status Error code defined in status\_t

Definition at line 4327 of file clock\_S32K1xx.c.

**16.28.6.8** `status_t CLOCK_SYS_Init ( clock_manager_user_config_t const ** clockConfigsPtr, uint8_t configsNumber, clock_manager_callback_user_config_t ** callbacksPtr, uint8_t callbacksNumber )`

Install pre-defined clock configurations.

This function installs the pre-defined clock configuration table to clock manager.



**Parameters**

in	<i>clockConfigsPtr</i>	Pointer to the clock configuration table.
in	<i>configsNumber</i>	Number of clock configurations in table.
in	<i>callbacksPtr</i>	Pointer to the callback configuration table.
in	<i>callbacks↔ Number</i>	Number of callback configurations in table.

**Returns**

Error code.

Definition at line 4140 of file clock\_S32K1xx.c.

#### 16.28.6.9 `status_t CLOCK_SYS_SetConfiguration ( clock_manager_user_config_t const * config )`

Set system clock configuration.

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**

in	<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. This function should be called only on run mode.

Definition at line 4339 of file clock\_S32K1xx.c.

#### 16.28.6.10 `status_t CLOCK_SYS_UpdateConfiguration ( uint8_t targetConfigIndex, clock_manager_policy_t policy )`

Set system clock configuration according to pre-defined structure.

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. The function should be called only on run mode.

**Parameters**

in	<i>targetConfig↔ Index</i>	Index of the clock configuration.
in	<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.

Definition at line 4173 of file clock\_S32K1xx.c.

### 16.28.7 Variable Documentation

#### 16.28.7.1 `uint32_t g_RtcClkInFreq`

RTC\_CLKIN clock frequency.

Definition at line 81 of file `clock_S32K1xx.c`.

#### 16.28.7.2 `uint32_t g_TClkFreq[NUMBER_OF_TCLK_INPUTS]`

TCLKx clocks

Definition at line 78 of file `clock_S32K1xx.c`.

#### 16.28.7.3 `uint32_t g_xtal0ClkFreq`

EXTAL0 clock frequency.

Definition at line 84 of file `clock_S32K1xx.c`.

#### 16.28.7.4 `const uint8_t peripheralFeaturesList[CLOCK_NAME_COUNT]`

Peripheral features list Constant array storing the mappings between clock names of the peripherals and feature lists.

Definition at line 432 of file `clock_S32K1xx.c`.

## 16.29 Common Core API.

### 16.29.1 Detailed Description

This group contains general core APIs that used for both protocol LIN 2.1 and J2602.

#### Modules

- [Driver and cluster management](#)  
*API perform the initialization of the LIN core.*
- [Interface management](#)  
*This group contains APIs that help users manage interface(s) in LIN node.*
- [Notification](#)  
*This group contains APIs that let users know when a signal's value changed.*
- [Schedule management](#)  
*This group contains APIs that help users manage schedule tables in master node only.*
- [Signal interaction](#)  
*This group contains APIs that help users interact with signals of LIN node.*
- [User provided call-outs](#)  
*This group contains APIs which may be called from within the LIN module in order to enable/disable LIN communication interrupts.*

#### Macros

- `#define` [SAVE\\_CONFIG\\_SET](#) 0x0040U
- `#define` [EVENT\\_TRIGGER\\_COLLISION\\_SET](#) 0x0020U
- `#define` [BUS\\_ACTIVITY\\_SET](#) 0x0010U
- `#define` [GO\\_TO\\_SLEEP\\_SET](#) 0x0008U
- `#define` [OVERRUN](#) 0x0004U
- `#define` [SUCCESSFULL\\_TRANSFER](#) 0x0002U
- `#define` [ERROR\\_IN\\_RESPONSE](#) 0x0001U

### 16.29.2 Macro Definition Documentation

#### 16.29.2.1 `#define` [BUS\\_ACTIVITY\\_SET](#) 0x0010U

Bus activity

Definition at line 32 of file `lin_common_api.h`.

#### 16.29.2.2 `#define` [ERROR\\_IN\\_RESPONSE](#) 0x0001U

Error in response

Definition at line 36 of file `lin_common_api.h`.

#### 16.29.2.3 `#define` [EVENT\\_TRIGGER\\_COLLISION\\_SET](#) 0x0020U

Event triggered frame collision

Definition at line 31 of file `lin_common_api.h`.

#### 16.29.2.4 `#define` [GO\\_TO\\_SLEEP\\_SET](#) 0x0008U

Go to sleep

Definition at line 33 of file `lin_common_api.h`.

**16.29.2.5 #define OVERRUN 0x0004U**

Overflow

Definition at line 34 of file lin\_common\_api.h.

**16.29.2.6 #define SAVE\_CONFIG\_SET 0x0040U**

Save configuration

Definition at line 30 of file lin\_common\_api.h.

**16.29.2.7 #define SUCCESSFULL\_TRANSFER 0x0002U**

Successful transfer

Definition at line 35 of file lin\_common\_api.h.

## 16.30 Common Transport Layer API

### 16.30.1 Detailed Description

Contains Transport Layer APIs that used for both protocols LIN 2.1 and J2602.

#### Modules

- [Cooked API](#)

*Cooked processing of diagnostic messages manages one complete message at a time.*

- [Initialization](#)

*Initialize transport layer (queues, status, ...).*

- [Raw API](#)

*The raw API is operating on PDU level and it is typically used to gateway PDUs between CAN and LIN.*

#### Macros

- `#define LD_READ_OK 0x33U`
- `#define LD_LENGTH_TOO_SHORT 0x34U`
- `#define LD_DATA_ERROR 0x43U`
- `#define LD_LENGTH_NOT_CORRECT 0x44U`
- `#define LD_SET_OK 0x45U`
- `#define SERVICE_TARGET_RESET 0xB5U`
- `#define RES_POSITIVE 0x40U`
- `#define LIN_PRODUCT_ID 0x00U`
- `#define LIN_SERIAL_NUMBER 0x01U`
- `#define LD_BROADCAST 0x7FU`
- `#define LD_FUNCTIONAL_NAD 0x7EU`
- `#define LD_ANY_SUPPLIER 0x7FFFU`
- `#define LD_ANY_FUNCTION 0xFFFFU`
- `#define LD_ANY_MESSAGE 0xFFFFU`
- `#define RES_NEGATIVE 0x7FU`
- `#define GENERAL_REJECT 0x10U`
- `#define SERVICE_NOT_SUPPORTED 0x11U`
- `#define SUBFUNCTION_NOT_SUPPORTED 0x12U`
- `#define NEGATIVE 0U`
- `#define POSITIVE 1U`
- `#define TRANSMITTING 0U`
- `#define RECEIVING 1U`
- `#define DIAG_SERVICE_CALLBACK_HANDLER(iii, sid) lin_diag_service_callback((iii), (sid))`

#### Functions

- void [lin\\_diag\\_service\\_callback](#) (l\_ifc\_handle iii, l\_u8 sid)

### 16.30.2 Macro Definition Documentation

16.30.2.1 `#define DIAG_SERVICE_CALLBACK_HANDLER( iii, sid ) lin_diag_service_callback((iii), (sid))`

Definition at line 86 of file `lin_commontl_api.h`.

**16.30.2.2 #define GENERAL\_REJECT 0x10U**

Error code raised when request for service not supported comes

Definition at line 71 of file lin\_commontl\_api.h.

**16.30.2.3 #define LD\_ANY\_FUNCTION 0xFFFFU**

Function

Definition at line 66 of file lin\_commontl\_api.h.

**16.30.2.4 #define LD\_ANY\_MESSAGE 0xFFFFU**

Message

Definition at line 67 of file lin\_commontl\_api.h.

**16.30.2.5 #define LD\_ANY\_SUPPLIER 0x7FFFU**

Supplier

Definition at line 65 of file lin\_commontl\_api.h.

**16.30.2.6 #define LD\_BROADCAST 0x7FU**

Broadcast NAD

Definition at line 63 of file lin\_commontl\_api.h.

**16.30.2.7 #define LD\_DATA\_ERROR 0x43U**

Data error

Definition at line 50 of file lin\_commontl\_api.h.

**16.30.2.8 #define LD\_FUNCTIONAL\_NAD 0x7EU**

Functional NAD

Definition at line 64 of file lin\_commontl\_api.h.

**16.30.2.9 #define LD\_LENGTH\_NOT\_CORRECT 0x44U**

Length not correct

Definition at line 51 of file lin\_commontl\_api.h.

**16.30.2.10 #define LD\_LENGTH\_TOO\_SHORT 0x34U**

Length too short

Definition at line 48 of file lin\_commontl\_api.h.

**16.30.2.11 #define LD\_READ\_OK 0x33U**

Read OK

Definition at line 47 of file lin\_commontl\_api.h.

**16.30.2.12 #define LD\_SET\_OK 0x45U**

Set OK

Definition at line 52 of file lin\_commontl\_api.h.

**16.30.2.13 #define LIN\_PRODUCT\_ID 0x00U**

Node product identifier

Definition at line 59 of file lin\_commontl\_api.h.

**16.30.2.14 #define LIN\_SERIAL\_NUMBER 0x01U**

Serial number

Definition at line 60 of file lin\_commontl\_api.h.

**16.30.2.15 #define NEGATIVE 0U**

Negative response

Definition at line 76 of file lin\_commontl\_api.h.

**16.30.2.16 #define POSITIVE 1U**

Positive response

Definition at line 77 of file lin\_commontl\_api.h.

**16.30.2.17 #define RECEIVING 1U**

Receiving

Definition at line 80 of file lin\_commontl\_api.h.

**16.30.2.18 #define RES\_NEGATIVE 0x7FU**

Negative response

Definition at line 70 of file lin\_commontl\_api.h.

**16.30.2.19 #define RES\_POSITIVE 0x40U**

Positive response

Definition at line 56 of file lin\_commontl\_api.h.

**16.30.2.20 #define SERVICE\_NOT\_SUPPORTED 0x11U**

Error code in negative response for not supported service

Definition at line 72 of file lin\_commontl\_api.h.

**16.30.2.21 #define SERVICE\_TARGET\_RESET 0xB5U**

Target reset service

Definition at line 55 of file lin\_commontl\_api.h.

**16.30.2.22 #define SUBFUNCTION\_NOT\_SUPPORTED 0x12U**

Error code in negative response for not supported sub function

Definition at line 73 of file lin\_commontl\_api.h.

**16.30.2.23 #define TRANSMITTING 0U**

Transmitting

Definition at line 79 of file lin\_commontl\_api.h.

### 16.30.3 Function Documentation

#### 16.30.3.1 void lin\_diag\_service\_callback ( l\_ifc\_handle *iii*, l\_u8 *sid* )

Definition at line 1041 of file lin\_diagnostic\_service.c.



## 16.31 Comparator (CMP)

### 16.31.1 Detailed Description

#### Hardware background

The comparator (CMP) module is an analog comparator integrated in MCU.

Features of the CMP module include:

- 8 bit DAC with 2 voltage reference source
- 8 analog inputs from external pins
- Round robin check. In summary, this allow the CMP to operate independently in STOP and VLPS mode, whilst being triggered periodically to sample up to 8 inputs. Only if an input changes state is a full wakeup generated.
- Operational over the entire supply range
- Inputs may range from rail to rail
- Programmable hysteresis control
- Selectable interrupt on rising-edge, falling-edge, or both rising or falling edges of the comparator output
- Selectable inversion on comparator output
- Capability to produce a wide range of outputs such as: sampled, windowed, which is ideal for certain PWM zero-crossing-detection applications and digitally filtered
- A comparison event can be selected to trigger a DMA transfer
- The window and filter functions are not available in STOP modes.

#### How to use the CMP driver in your application

The user can configure the CMP in many ways: -CMP\_DRV\_Init - configures all CMP features -CMP\_DRV\_↔ ConfigDAC - configures only DAC features -CMP\_DRV\_ConfigTriggerMode - configures only trigger mode features -CMP\_DRV\_ConfigComparator - configures only analog comparator features -CMP\_DRV\_ConfigMUX - configures only MUX features

Also the current configuration can be read using: -CMP\_DRV\_GetConfigAll - gets all CMP configuration -CM↔ P\_DRV\_GetDACConfig - gets only DAC configuration -CMP\_DRV\_GetMUXConfig - gets only MUX configuration -CMP\_DRV\_GetInitTriggerMode - gets only trigger mode configuration -CMP\_DRV\_GetComparatorConfig - gets only analog comparator features

A default configuration can be read using: -CMP\_DRV\_GetDefaultConfig - gets a default configuration for the comparator

When the MCU exits from STOP mode CMP\_DRV\_GetInputFlags can be used to get the channel which triggered the wakeup. Please use this function only in this use case. CMP\_DRV\_ClearInputFlags will be used to clear this input change flags.

CMP\_DRV\_GetOutputFlags can be used to get output flag state and CMP\_DRV\_ClearOutputFlags to clear them.

The main structure used to configure your application is [cmp\\_module\\_t](#). This structure includes configuration structures for trigger mode, MUX, DAC and comparator: [cmp\\_comparator\\_t](#), [cmp\\_anmux\\_t](#), [cmp\\_dac\\_t](#) and [cmp\\_trigger\\_mode\\_t](#)

If application use CMP as wakeup source from Standby mode on MPC574x devices is mandatory to enable channel 3

## Integration guideline ##

## Compilation units

The following files need to be compiled in the project:

```
{S32SDK_PATH}\platform\drivers\src\cmp\cmp_driver.c
{S32SDK_PATH}\platform\drivers\src\cmp\cmp_hw_access.c
```

## Include path

The following paths need to be added to the include path of the toolchain:

```
{S32SDK_PATH}\platform\drivers\inc\
{S32SDK_PATH}\platform\drivers\src\cmp\
```

## Compile symbols

No special symbols are required for this component

## Dependencies

[Clock Manager Interrupt Manager \(Interrupt\) PinSettings](#)

## Example for S32K14x:

The next example will compare 2 external signals (CMP input 0 an CMP input 1). The output can be measured on port E, pin 4.

```
const cmp_module_t cmp_general_config =
{
    {
        .dmaTriggerState      = false,
        .outputInterruptTrigger = CMP_NO_EVENT,
        .mode                 = CMP_CONTINUOUS,
        .filterSamplePeriod   = 0,
        .filterSampleCount    = 0,
        .powerMode            = CMP_LOW_SPEED,
        .inverterState        = CMP_NORMAL,
        .outputSelect         = CMP_COUT,
        .pinState             = CMP_AVAILABLE,
        .offsetLevel          = CMP_LEVEL_OFFSET_0,
        .hysteresisLevel      = CMP_LEVEL_HYS_0
    },
    {
        .positivePortMux      = CMP_MUX,
        .negativePortMux      = CMP_MUX,
        .positiveInputMux     = 0,
        .negativeInputMux     = 1
    },
    {
        .voltageReferenceSource = CMP_VIN1,
        .voltage                = 120,
        .state                  = false
    },
    {
        .roundRobinState      = false,
        .roundRobinInterruptState = false,
        .fixedPort            = CMP_PLUS_FIXED,
        .fixedChannel         = 0,
        .samples              = 0,
        .initializationDelay   = 0,
        /* Channel 0 is enabled for round robin check */
        /* Channel 1 is enabled for round robin check */
        /* Channel 2 is enabled for round robin check */
        /* Channel 3 is enabled for round robin check */
        /* Channel 4 is enabled for round robin check */
        /* Channel 5 is enabled for round robin check */
        /* Channel 6 is enabled for round robin check */
        /* Channel 7 is enabled for round robin check */
        .roundRobinChannelsState = 255,
        /* Initial comparison result for channel 0 is 1 */
        /* Initial comparison result for channel 1 is 1 */
        /* Initial comparison result for channel 2 is 1 */
        /* Initial comparison result for channel 3 is 1 */
        /* Initial comparison result for channel 4 is 1 */
    }
}
```

```

        /* Initial comparison result for channel 5 is 1 */
        /* Initial comparison result for channel 6 is 1 */
        /* Initial comparison result for channel 7 is 1 */
        .programedState      = 255
    }
};

#define COMPARATOR_PORT      PORTA
#define COMPARATOR_INPUT1_PIN 0UL
#define COMPARATOR_INPUT2_PIN 1UL
#define COMPARATOR_OUTPUT    4UL
#define COMPARATOR_INSTANCE  0UL

int main(void)
{
    /* Initialize and configure clocks
     * - Setup system clocks
     * - Enable clock feed for Ports and Comparator
     * - See Clock Manager component for more info
     */
    CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
                  g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
    CLOCK_SYS_UpdateConfiguration(0U,
                                  CLOCK_MANAGER_POLICY_AGREEMENT);

    /* Set pins used by CMP */
    /* The negative port is connected to PTA0 and positive port is connected to PTA1. The
     * comparator output can be visualized on PTA4 */
    /* Initialize pins
     * - Setup input pins for Comparator
     * - Setup output pins for LEDs
     * - See PinSettings component for more info
     */
    PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);

    /* Init CMP module */
    CMP_DRV_Init(COMPARATOR_INSTANCE, &cmp_general_config);
    for (;;)
    {
    }
    return(0);
}

```

#### Example for MPC574XG:

The next example will compare 2 external signals (CMP input 0 an CMP input 1). The output can be measured on port E, pin 4.

```

const cmp_module_t cmp_general_config =
{
    {
        .dmaTriggerState      = false,
        .outputInterruptTrigger = CMP_NO_EVENT,
        .mode                  = CMP_CONTINUOUS,
        .filterSamplePeriod    = 0,
        .filterSampleCount     = 0,
        .powerMode              = CMP_LOW_SPEED,
        .inverterState          = CMP_NORMAL,
        .outputSelect           = CMP_COUT,
        .pinState                = CMP_AVAILABLE,
        .hysteresisLevel        = CMP_LEVEL_HYS_0
    },
    {
        .positivePortMux       = CMP_MUX,
        .negativePortMux        = CMP_MUX,
        .positiveInputMux       = 0,
        .negativeInputMux       = 1
    },
    {
        .voltageReferenceSource = CMP_VIN1,
        .voltage                  = 120,
        .state                     = false,
        .fixRefInputMux            = false
    },
    {
        .roundRobinState        = false,
        .roundRobinInterruptState = false,
        .fixedPort                = CMP_PLUS_FIXED,
        .fixedChannel             = 0,
        .samples                  = 0,
        /* Channel 0 is enabled for round robin check */
        /* Channel 1 is enabled for round robin check */
    }
}

```

```

        /* Channel 2 is enabled for round robin check */
        /* Channel 3 is enabled for round robin check */
        /* Channel 4 is enabled for round robin check */
        /* Channel 5 is enabled for round robin check */
        /* Channel 6 is enabled for round robin check */
        /* Channel 7 is enabled for round robin check */
        .roundRobinChannelsState = 255,
        /* Initial comparison result for channel 0 is 1 */
        /* Initial comparison result for channel 1 is 1 */
        /* Initial comparison result for channel 2 is 1 */
        /* Initial comparison result for channel 3 is 1 */
        /* Initial comparison result for channel 4 is 1 */
        /* Initial comparison result for channel 5 is 1 */
        /* Initial comparison result for channel 6 is 1 */
        /* Initial comparison result for channel 7 is 1 */
        .programedState = 255
    }
};

#define COMPARATOR_PORT          PORTA
#define COMPARATOR_INPUT1_PIN    0UL
#define COMPARATOR_INPUT2_PIN    1UL
#define COMPARATOR_OUTPUT        4UL
#define COMPARATOR_INSTANCE      0UL

int main(void)
{
    /* Write your local variable definition here */
    /* Initialize and configure clocks
     * - Setup system clocks
     * - Enable clock feed for Ports and Comparator
     * - See Clock Manager component for more info
     */
    CLOCK_SYS_Init(g_clockManConfigsArr, CLOCK_MANAGER_CONFIG_CNT,
                  g_clockManCallbacksArr, CLOCK_MANAGER_CALLBACK_CNT);
    CLOCK_SYS_UpdateConfiguration(0U,
    CLOCK_MANAGER_POLICY_AGREEMENT);

    /* Set pins used by CMP */
    /* The negative port is connected to PTA0 and positive port is connected to PTA1. The
    comparator output can be visualized on PTA4 */
    /* Initialize pins
     * - Setup input pins for Comparator
     * - Setup output pins for LEDs
     * - See PinSettings component for more info
     */
    PINS_DRV_Init(NUM_OF_CONFIGURED_PINS, g_pin_mux_InitConfigArr);

    /* Init CMP module */
    CMP_DRV_Init(COMPARATOR_INSTANCE, &cmp_general_config);
    for (;;)
    {
        return(0);
    }
}

```

## Modules

- [Comparator Driver](#)

*Comparator Peripheral Driver.*

## 16.32 Comparator Driver

### 16.32.1 Detailed Description

Comparator Peripheral Driver.

Definitions

#### Data Structures

- struct [cmp\\_comparator\\_t](#)  
*Defines the block configuration. [More...](#)*
- struct [cmp\\_anmux\\_t](#)  
*Defines the analog mux. [More...](#)*
- struct [cmp\\_dac\\_t](#)  
*Defines the DAC block. [More...](#)*
- struct [cmp\\_trigger\\_mode\\_t](#)  
*Defines the trigger mode. [More...](#)*
- struct [cmp\\_module\\_t](#)  
*Defines the comparator module configuration. [More...](#)*

#### Macros

- #define [CMP\\_INPUT\\_FLAGS\\_MASK](#) 0xFF0000
- #define [CMP\\_INPUT\\_FLAGS\\_SHIFT](#) 16U
- #define [CMP\\_ROUND\\_ROBIN\\_CHANNELS\\_MASK](#) 0xFF0000
- #define [CMP\\_ROUND\\_ROBIN\\_CHANNELS\\_SHIFT](#) 16U

#### Typedefs

- typedef uint8\_t [cmp\\_ch\\_list\\_t](#)  
*Comparator channels list (1bit/channel) |-----|-----|---|-----|-----| |CH7\_state|CH6\_state|.....|CH1\_↔  
state|CH0\_state| |-----|-----|---|-----|-----| Implements : cmp\_ch\_list\_t\_Class.*
- typedef uint8\_t [cmp\\_ch\\_number\\_t](#)  
*Number of channel Implements : cmp\_ch\_number\_t\_Class.*

#### Enumerations

- enum [cmp\\_power\\_mode\\_t](#) { [CMP\\_LOW\\_SPEED](#) = 0U, [CMP\\_HIGH\\_SPEED](#) = 1U }  
*Power Modes selection Implements : cmp\_power\_mode\_t\_Class.*
- enum [cmp\\_voltage\\_reference\\_t](#) { [CMP\\_VIN1](#) = 0U, [CMP\\_VIN2](#) = 1U }  
*Voltage Reference selection Implements : cmp\_voltage\_reference\_t\_Class.*
- enum [cmp\\_port\\_mux\\_t](#) { [CMP\\_DAC](#) = [CMP\\_DAC\\_SOURCE](#), [CMP\\_MUX](#) = [CMP\\_MUX\\_SOURCE](#) }  
*Port Mux Source selection Implements : cmp\_port\_mux\_t\_Class.*
- enum [cmp\\_inverter\\_t](#) { [CMP\\_NORMAL](#) = 0U, [CMP\\_INVERT](#) = 1U }  
*Comparator output invert selection Implements : cmp\_inverter\_t\_Class.*
- enum [cmp\\_output\\_select\\_t](#) { [CMP\\_COUT](#) = 0U, [CMP\\_COUTA](#) = 1U }  
*Comparator output select selection Implements : cmp\_output\_select\_t\_Class.*
- enum [cmp\\_output\\_enable\\_t](#) { [CMP\\_UNAVAILABLE](#) = 0U, [CMP\\_AVAILABLE](#) = 1U }  
*Comparator output pin enable selection Implements : cmp\_output\_enable\_t\_Class.*
- enum [cmp\\_hysteresis\\_t](#) { [CMP\\_LEVEL\\_HYS\\_0](#) = 0U, [CMP\\_LEVEL\\_HYS\\_1](#) = 1U, [CMP\\_LEVEL\\_HYS\\_2](#) = 2U, [CMP\\_LEVEL\\_HYS\\_3](#) = 3U }

Comparator hysteresis control Implements : *cmp\_hysteresis\_t\_Class*.

- enum `cmp_fixed_port_t` { `CMP_PLUS_FIXED` = 0U, `CMP_MINUS_FIXED` = 1U }

Comparator Round-Robin fixed port Implements : *cmp\_fixed\_port\_t\_Class*.

- enum `cmp_output_trigger_t` { `CMP_NO_EVENT` = 0U, `CMP_FALLING_EDGE` = 1U, `CMP_RISING_EDGE` = 2U, `CMP_BOTH_EDGES` = 3U }

Comparator output interrupt configuration Implements : *cmp\_output\_trigger\_t\_Class*.

- enum `cmp_mode_t` {  
`CMP_DISABLED` = 0U, `CMP_CONTINUOUS` = 1U, `CMP_SAMPLED_NONFILTRED_INT_CLK` = 2U, `CMP_SAMPLED_NONFILTRED_EXT_CLK` = 3U,  
`CMP_SAMPLED_FILTRED_INT_CLK` = 4U, `CMP_SAMPLED_FILTRED_EXT_CLK` = 5U, `CMP_WINDOWED_RESAMPLED` = 6U, `CMP_WINDOWED_FILTRED` = 7U,  
`CMP_WINDOWED_FILTRED` = 8U }

Comparator functional modes Implements : *cmp\_mode\_t\_Class*.

## cMP DRV.

- status\_t `CMP_DRV_Reset` (const uint32\_t instance)  
Reset all registers.
- status\_t `CMP_DRV_GetInitConfigAll` (cmp\_module\_t \*config)  
Get reset configuration for all registers.
- status\_t `CMP_DRV_GetDefaultConfig` (cmp\_module\_t \*const config)  
Gets a default comparator configuration.
- status\_t `CMP_DRV_Init` (const uint32\_t instance, const cmp\_module\_t \*const config)  
Configure all comparator features with the given configuration structure.
- status\_t `CMP_DRV_GetConfigAll` (const uint32\_t instance, cmp\_module\_t \*const config)  
Gets the current comparator configuration.
- status\_t `CMP_DRV_GetInitConfigDAC` (cmp\_dac\_t \*config)  
Get reset configuration for registers related with DAC.
- status\_t `CMP_DRV_ConfigDAC` (const uint32\_t instance, const cmp\_dac\_t \*config)  
Configure only the DAC component.
- status\_t `CMP_DRV_GetDACConfig` (const uint32\_t instance, cmp\_dac\_t \*const config)  
Return current configuration for DAC.
- status\_t `CMP_DRV_GetInitConfigMUX` (cmp\_anmux\_t \*config)  
Get reset configuration for registers related with MUX.
- status\_t `CMP_DRV_ConfigMUX` (const uint32\_t instance, const cmp\_anmux\_t \*config)  
Configure only the MUX component.
- status\_t `CMP_DRV_GetMUXConfig` (const uint32\_t instance, cmp\_anmux\_t \*const config)  
Return configuration only for the MUX component.
- status\_t `CMP_DRV_GetInitTriggerMode` (cmp\_trigger\_mode\_t \*config)  
Get reset configuration for registers related with Trigger Mode.
- status\_t `CMP_DRV_ConfigTriggerMode` (const uint32\_t instance, const cmp\_trigger\_mode\_t \*config)  
Configure trigger mode.
- status\_t `CMP_DRV_GetTriggerModeConfig` (const uint32\_t instance, cmp\_trigger\_mode\_t \*const config)  
Get current trigger mode configuration.
- status\_t `CMP_DRV_GetOutputFlags` (const uint32\_t instance, cmp\_output\_trigger\_t \*flags)  
Get comparator output flags.
- status\_t `CMP_DRV_ClearOutputFlags` (const uint32\_t instance)  
Clear comparator output flags.
- status\_t `CMP_DRV_GetInputFlags` (const uint32\_t instance, cmp\_ch\_list\_t \*flags)  
Gets input channels change flags.
- status\_t `CMP_DRV_ClearInputFlags` (const uint32\_t instance)

*Clear comparator input channels flags.*

- status\_t [CMP\\_DRV\\_GetInitConfigComparator](#) (cmp\_comparator\_t \*config)

*Get reset configuration for registers related with comparator features.*

- status\_t [CMP\\_DRV\\_ConfigComparator](#) (const uint32\_t instance, const cmp\_comparator\_t \*config)

*Configure only comparator features.*

- status\_t [CMP\\_DRV\\_GetComparatorConfig](#) (const uint32\_t instance, cmp\_comparator\_t \*config)

*Return configuration for comparator from CMP module.*

## 16.32.2 Data Structure Documentation

### 16.32.2.1 struct cmp\_comparator\_t

Defines the block configuration.

This structure is used to configure only comparator block module(filtering, sampling, power\_mode etc.) Implements : cmp\_comparator\_t\_Class

Definition at line 178 of file cmp\_driver.h.

#### Data Fields

- bool [dmaTriggerState](#)
- [cmp\\_output\\_trigger\\_t](#) outputInterruptTrigger
- [cmp\\_mode\\_t](#) mode
- uint8\_t [filterSamplePeriod](#)
- uint8\_t [filterSampleCount](#)
- [cmp\\_power\\_mode\\_t](#) powerMode
- [cmp\\_inverter\\_t](#) inverterState
- [cmp\\_output\\_enable\\_t](#) pinState
- [cmp\\_output\\_select\\_t](#) outputSelect
- [cmp\\_hysteresis\\_t](#) hysteresisLevel

#### Field Documentation

##### 16.32.2.1.1 bool dmaTriggerState

True if DMA transfer trigger from comparator is enable.

Definition at line 180 of file cmp\_driver.h.

##### 16.32.2.1.2 uint8\_t filterSampleCount

Number of sample count for filtering.

Definition at line 187 of file cmp\_driver.h.

##### 16.32.2.1.3 uint8\_t filterSamplePeriod

Filter sample period.

Definition at line 186 of file cmp\_driver.h.

##### 16.32.2.1.4 cmp\_hysteresis\_t hysteresisLevel

CMP\_LEVEL\_HYS\_0 if hard block output has level 0 hysteresis. CMP\_LEVEL\_HYS\_1 if hard block output has level 1 hysteresis. CMP\_LEVEL\_HYS\_2 if hard block output has level 2 hysteresis. CMP\_LEVEL\_HYS\_3 if hard block output has level 3 hysteresis.

Definition at line 200 of file cmp\_driver.h.

#### 16.32.2.1.5 `cmp_inverter_t` `inverterState`

CMP\_NORMAL if does not invert the comparator output. CMP\_INVERT if inverts the comparator output.

Definition at line 190 of file `cmp_driver.h`.

#### 16.32.2.1.6 `cmp_mode_t` `mode`

Configuration structure which define: the comparator functional mode, sample period and sample count.

Definition at line 185 of file `cmp_driver.h`.

#### 16.32.2.1.7 `cmp_output_trigger_t` `outputInterruptTrigger`

CMP\_NO\_INTERRUPT comparator output would not trigger any interrupt. CMP\_FALLING\_EDGE comparator output would trigger an interrupt on falling edge. CMP\_RISING\_EDGE comparator output would trigger an interrupt on rising edge. CMP\_BOTH\_EDGES comparator output would trigger an interrupt on rising and falling edges.

Definition at line 181 of file `cmp_driver.h`.

#### 16.32.2.1.8 `cmp_output_select_t` `outputSelect`

CMP\_COUT if output signal is equal to COUT(filtered). CMP\_COUTA if output signal is equal to COUTA(unfiltered).

Definition at line 194 of file `cmp_driver.h`.

#### 16.32.2.1.9 `cmp_output_enable_t` `pinState`

CMP\_UNAVAILABLE if comparator output is not available to package pin. CMP\_AVAILABLE if comparator output is available to package pin.

Definition at line 192 of file `cmp_driver.h`.

#### 16.32.2.1.10 `cmp_power_mode_t` `powerMode`

CMP\_LOW\_SPEED if low speed mode is selected. CMP\_HIGH\_SPEED if high speed mode is selected

Definition at line 188 of file `cmp_driver.h`.

#### 16.32.2.2 `struct cmp_anmux_t`

Defines the analog mux.

This structure is used to configure the analog multiplexor to select compared signals Implements : `cmp_anmux_t` ↔ `t_Class`

Definition at line 212 of file `cmp_driver.h`.

#### Data Fields

- [cmp\\_port\\_mux\\_t](#) `positivePortMux`
- [cmp\\_port\\_mux\\_t](#) `negativePortMux`
- [cmp\\_ch\\_number\\_t](#) `positiveInputMux`
- [cmp\\_ch\\_number\\_t](#) `negativeInputMux`

#### Field Documentation

##### 16.32.2.2.1 `cmp_ch_number_t` `negativeInputMux`

Select which channel is selected for the minus mux.

Definition at line 222 of file `cmp_driver.h`.

##### 16.32.2.2.2 `cmp_port_mux_t` `negativePortMux`

Select negative port signal. CMP\_DAC if source is digital to analog converter. CMP\_MUX if source is 8 ch MUX



Definition at line 217 of file cmp\_driver.h.

#### 16.32.2.2.3 `cmp_ch_number_t` positiveInputMux

Select which channel is selected for the plus mux.

Definition at line 221 of file cmp\_driver.h.

#### 16.32.2.2.4 `cmp_port_mux_t` positivePortMux

Select positive port signal. CMP\_DAC if source is digital to analog converter. CMP\_MUX if source is 8 ch MUX

Definition at line 214 of file cmp\_driver.h.

#### 16.32.2.3 `struct cmp_dac_t`

Defines the DAC block.

This structure is used to configure the DAC block integrated in comparator module Implements : `cmp_dac_t_Class`

Definition at line 231 of file cmp\_driver.h.

##### Data Fields

- `cmp_voltage_reference_t` voltageReferenceSource
- `uint8_t` voltage
- `bool` state

##### Field Documentation

#### 16.32.2.3.1 `bool` state

True if DAC is enabled.

Definition at line 236 of file cmp\_driver.h.

#### 16.32.2.3.2 `uint8_t` voltage

The digital value which is converted to analog signal.

Definition at line 235 of file cmp\_driver.h.

#### 16.32.2.3.3 `cmp_voltage_reference_t` voltageReferenceSource

CMP\_VIN1 if selected voltage reference is VIN1. CMP\_VIN2 if selected voltage reference is VIN2.

Definition at line 233 of file cmp\_driver.h.

#### 16.32.2.4 `struct cmp_trigger_mode_t`

Defines the trigger mode.

This structure is used to configure the trigger mode operation when MCU enters STOP modes Implements : `cmp_trigger_mode_t_Class`

Definition at line 251 of file cmp\_driver.h.

##### Data Fields

- `bool` roundRobinState
- `bool` roundRobinInterruptState
- `cmp_fixed_port_t` fixedPort
- `cmp_ch_number_t` fixedChannel
- `uint8_t` samples
- `cmp_ch_list_t` roundRobinChannelsState
- `cmp_ch_list_t` programmedState

## Field Documentation

### 16.32.2.4.1 `cmp_ch_number_t` fixedChannel

Select which channel would be assigned to the fixed port.

Definition at line 257 of file `cmp_driver.h`.

### 16.32.2.4.2 `cmp_fixed_port_t` fixedPort

CMP\_PLUS\_FIXED if plus port is fixed. CMP\_MINUS\_FIXED if minus port is fixed.

Definition at line 255 of file `cmp_driver.h`.

### 16.32.2.4.3 `cmp_ch_list_t` progradedState

Pre-programmed state for comparison result.

Definition at line 266 of file `cmp_driver.h`.

### 16.32.2.4.4 `cmp_ch_list_t` roundRobinChannelsState

One bite for each channel state. |CH7\_state|CH6\_state|.....|CH1\_state|CH0\_state|

Definition at line 262 of file `cmp_driver.h`.

### 16.32.2.4.5 `bool` roundRobinInterruptState

True if Round-Robin interrupt is enabled.

Definition at line 254 of file `cmp_driver.h`.

### 16.32.2.4.6 `bool` roundRobinState

True if Round-Robin is enabled.

Definition at line 253 of file `cmp_driver.h`.

### 16.32.2.4.7 `uint8_t` samples

Select number of round-robin clock cycles for a given channel.

Definition at line 258 of file `cmp_driver.h`.

### 16.32.2.5 `struct cmp_module_t`

Defines the comparator module configuration.

This structure is used to configure all components of comparator module Implements : `cmp_module_t_Class`

Definition at line 275 of file `cmp_driver.h`.

## Data Fields

- [cmp\\_comparator\\_t](#) comparator
- [cmp\\_anmux\\_t](#) mux
- [cmp\\_dac\\_t](#) dac
- [cmp\\_trigger\\_mode\\_t](#) triggerMode

## Field Documentation

### 16.32.2.5.1 `cmp_comparator_t` comparator

Definition at line 277 of file `cmp_driver.h`.

16.32.2.5.2 `cmp_dac_t` `dac`

Definition at line 279 of file `cmp_driver.h`.

16.32.2.5.3 `cmp_anmux_t` `mux`

Definition at line 278 of file `cmp_driver.h`.

16.32.2.5.4 `cmp_trigger_mode_t` `triggerMode`

Definition at line 280 of file `cmp_driver.h`.

## 16.32.3 Macro Definition Documentation

16.32.3.1 `#define CMP_INPUT_FLAGS_MASK 0xFF0000`

Definition at line 39 of file `cmp_driver.h`.

16.32.3.2 `#define CMP_INPUT_FLAGS_SHIFT 16U`

Definition at line 40 of file `cmp_driver.h`.

16.32.3.3 `#define CMP_ROUND_ROBIN_CHANNELS_MASK 0xFF0000`

Definition at line 41 of file `cmp_driver.h`.

16.32.3.4 `#define CMP_ROUND_ROBIN_CHANNELS_SHIFT 16U`

Definition at line 42 of file `cmp_driver.h`.

## 16.32.4 Typedef Documentation

16.32.4.1 `typedef uint8_t cmp_ch_list_t`

Comparator channels list (1bit/channel) |-----|-----|---|-----|-----| |CH7\_state|CH6\_state|.....|CH1\_↔  
state|CH0\_state| |-----|-----|---|-----|-----| Implements : `cmp_ch_list_t_Class`.

Definition at line 165 of file `cmp_driver.h`.

16.32.4.2 `typedef uint8_t cmp_ch_number_t`

Number of channel Implements : `cmp_ch_number_t_Class`.

Definition at line 170 of file `cmp_driver.h`.

## 16.32.5 Enumeration Type Documentation

16.32.5.1 `enum cmp_fixed_port_t`

Comparator Round-Robin fixed port Implements : `cmp_fixed_port_t_Class`.

## Enumerator

***CMP\_PLUS\_FIXED*** The Plus port is fixed. Only the inputs to the Minus port are swept in each round.

***CMP\_MINUS\_FIXED*** The Minus port is fixed. Only the inputs to the Plus port are swept in each round.

Definition at line 126 of file `cmp_driver.h`.

16.32.5.2 enum **cmp\_hysteresis\_t**

Comparator hysteresis control Implements : **cmp\_hysteresis\_t\_Class**.

Enumerator

**CMP\_LEVEL\_HYS\_0**

**CMP\_LEVEL\_HYS\_1**

**CMP\_LEVEL\_HYS\_2**

**CMP\_LEVEL\_HYS\_3**

Definition at line 115 of file **cmp\_driver.h**.

16.32.5.3 enum **cmp\_inverter\_t**

Comparator output invert selection Implements : **cmp\_inverter\_t\_Class**.

Enumerator

**CMP\_NORMAL** Output signal isn't inverted.

**CMP\_INVERT** Output signal is inverted.

Definition at line 77 of file **cmp\_driver.h**.

16.32.5.4 enum **cmp\_mode\_t**

Comparator functional modes Implements : **cmp\_mode\_t\_Class**.

Enumerator

**CMP\_DISABLED**

**CMP\_CONTINUOUS**

**CMP\_SAMPLED\_NONFILTRED\_INT\_CLK**

**CMP\_SAMPLED\_NONFILTRED\_EXT\_CLK**

**CMP\_SAMPLED\_FILTRED\_INT\_CLK**

**CMP\_SAMPLED\_FILTRED\_EXT\_CLK**

**CMP\_WINDOWED**

**CMP\_WINDOWED\_RESAMPLED**

**CMP\_WINDOWED\_FILTRED**

Definition at line 146 of file **cmp\_driver.h**.

16.32.5.5 enum **cmp\_output\_enable\_t**

Comparator output pin enable selection Implements : **cmp\_output\_enable\_t\_Class**.

Enumerator

**CMP\_UNAVAILABLE** Comparator output isn't available to a specific pin

**CMP\_AVAILABLE** Comparator output is available to a specific pin

Definition at line 95 of file **cmp\_driver.h**.

16.32.5.6 enum **cmp\_output\_select\_t**

Comparator output select selection Implements : **cmp\_output\_select\_t\_Class**.

## Enumerator

**CMP\_COUT** Select COUT as comparator output signal.

**CMP\_COUTA** Select COUTA as comparator output signal.

Definition at line 86 of file **cmp\_driver.h**.

16.32.5.7 enum **cmp\_output\_trigger\_t**

Comparator output interrupt configuration Implements : **cmp\_output\_trigger\_t\_Class**.

## Enumerator

**CMP\_NO\_EVENT** Comparator output interrupts are disabled OR no event occurred.

**CMP\_FALLING\_EDGE** Comparator output interrupts will be generated only on falling edge OR only falling edge event occurred.

**CMP\_RISING\_EDGE** Comparator output interrupts will be generated only on rising edge OR only rising edge event occurred.

**CMP\_BOTH\_EDGES** Comparator output interrupts will be generated on both edges OR both edges event occurred.

Definition at line 135 of file **cmp\_driver.h**.

16.32.5.8 enum **cmp\_port\_mux\_t**

Port Mux Source selection Implements : **cmp\_port\_mux\_t\_Class**.

## Enumerator

**CMP\_DAC** Select DAC as source for the comparator port.

**CMP\_MUX** Select MUX8 as source for the comparator port.

Definition at line 68 of file **cmp\_driver.h**.

16.32.5.9 enum **cmp\_power\_mode\_t**

Power Modes selection Implements : **cmp\_power\_mode\_t\_Class**.

## Enumerator

**CMP\_LOW\_SPEED** Module in low speed mode.

**CMP\_HIGH\_SPEED** Module in high speed mode.

Definition at line 50 of file **cmp\_driver.h**.

16.32.5.10 enum **cmp\_voltage\_reference\_t**

Voltage Reference selection Implements : **cmp\_voltage\_reference\_t\_Class**.

## Enumerator

**CMP\_VIN1** Use Vin1 as supply reference source for DAC.

**CMP\_VIN2** Use Vin2 as supply reference source for DAC.

Definition at line 59 of file **cmp\_driver.h**.

### 16.32.6 Function Documentation

#### 16.32.6.1 `status_t CMP_DRV_ClearInputFlags ( const uint32_t instance )`

Clear comparator input channels flags.

This function clear comparator input channels flags.

##### Parameters

<i>instance</i>	- instance number
-----------------	-------------------

##### Returns

- `STATUS_SUCCESS` : Completed successfully.
- `STATUS_ERROR` : Error occurred.

Definition at line 562 of file `cmp_driver.c`.

#### 16.32.6.2 `status_t CMP_DRV_ClearOutputFlags ( const uint32_t instance )`

Clear comparator output flags.

This function clear comparator output flags(rising and falling edge).

##### Parameters

<i>instance</i>	- instance number
-----------------	-------------------

##### Returns

- `STATUS_SUCCESS` : Completed successfully.
- `STATUS_ERROR` : Error occurred.

Definition at line 517 of file `cmp_driver.c`.

#### 16.32.6.3 `status_t CMP_DRV_ConfigComparator ( const uint32_t instance, const cmp_comparator_t * config )`

Configure only comparator features.

This function configure only features related with comparator: DMA request, power mode, output select, interrupts enable, invert, offset, hysteresis.

##### Parameters

<i>instance</i>	- instance number
<i>config</i>	- the configuration structure

##### Returns

- `STATUS_SUCCESS` : Completed successfully.
- `STATUS_ERROR` : Error occurred.

Definition at line 611 of file `cmp_driver.c`.

#### 16.32.6.4 `status_t CMP_DRV_ConfigDAC ( const uint32_t instance, const cmp_dac_t * config )`

Configure only the DAC component.

This function configures the DAC with the options provided in the config structure.

## Parameters

<i>instance</i>	- instance number
<i>config</i>	- the configuration structure

## Returns

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 316 of file cmp\_driver.c.

**16.32.6.5** `status_t CMP_DRV_ConfigMUX ( const uint32_t instance, const cmp_anmux_t * config )`

Configure only the MUX component.

This function configures the MUX with the options provided in the config structure.

## Parameters

<i>instance</i>	- instance number
<i>config</i>	- the configuration structure

## Returns

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 381 of file cmp\_driver.c.

**16.32.6.6** `status_t CMP_DRV_ConfigTriggerMode ( const uint32_t instance, const cmp_trigger_mode_t * config )`

Configure trigger mode.

This function configures the trigger mode with the options provided in the config structure.

## Parameters

<i>instance</i>	- instance number
<i>config</i>	- the configuration structure

## Returns

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 445 of file cmp\_driver.c.

**16.32.6.7** `status_t CMP_DRV_GetComparatorConfig ( const uint32_t instance, cmp_comparator_t * config )`

Return configuration for comparator from CMP module.

This function return configuration for features related with comparator: DMA request, power mode, output select, interrupts enable, invert, offset, hysteresis.

## Parameters

<i>instance</i>	- instance number
-----------------	-------------------

<i>config</i>	- the configuration structure returned
---------------	--

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 641 of file cmp\_driver.c.

#### 16.32.6.8 status\_t CMP\_DRV\_GetConfigAll ( const uint32\_t instance, cmp\_module\_t \*const config )

Gets the current comparator configuration.

This function returns the current configuration for comparator as a configuration structure.

**Parameters**

<i>instance</i>	- instance number
<i>config</i>	- the configuration structure

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 242 of file cmp\_driver.c.

#### 16.32.6.9 status\_t CMP\_DRV\_GetDACConfig ( const uint32\_t instance, cmp\_dac\_t \*const config )

Return current configuration for DAC.

This function returns current configuration only for DAC.

**Parameters**

<i>instance</i>	- instance number
<i>config</i>	- the configuration structure

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 340 of file cmp\_driver.c.

#### 16.32.6.10 status\_t CMP\_DRV\_GetDefaultConfig ( cmp\_module\_t \*const config )

Gets a default comparator configuration.

This function returns a default configuration for the comparator as a configuration structure.

**Parameters**

<i>instance</i>	- instance number
<i>config</i>	- the configuration structure

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 137 of file cmp\_driver.c.



**16.32.6.11 status\_t CMP\_DRV\_GetInitConfigAll ( cmp\_module\_t \* config )**

Get reset configuration for all registers.

This function returns a configuration structure with reset values for all registers from comparator module.

**Parameters**

<i>config</i>	- the configuration structure
---------------	-------------------------------

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 85 of file cmp\_driver.c.

**16.32.6.12 status\_t CMP\_DRV\_GetInitConfigComparator ( cmp\_comparator\_t \* config )**

Get reset configuration for registers related with comparator features.

This function return a configuration structure with reset values for features associated with comparator (DMA request, power mode, output select, interrupts enable, invert, offset, hysteresis).

**Parameters**

<i>config</i>	- the configuration structure
---------------	-------------------------------

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 584 of file cmp\_driver.c.

**16.32.6.13 status\_t CMP\_DRV\_GetInitConfigDAC ( cmp\_dac\_t \* config )**

Get reset configuration for registers related with DAC.

This function returns a configuration structure with reset values for features associated with DAC.

**Parameters**

<i>config</i>	- the configuration structure
---------------	-------------------------------

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 295 of file cmp\_driver.c.

**16.32.6.14 status\_t CMP\_DRV\_GetInitConfigMUX ( cmp\_anmux\_t \* config )**

Get reset configuration for registers related with MUX.

This function returns a configuration structure with reset values for features associated with MUX.

**Parameters**

<i>config</i>	- the configuration structure
---------------	-------------------------------

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 362 of file cmp\_driver.c.

**16.32.6.15** status\_t CMP\_DRV\_GetInitTriggerMode ( cmp\_trigger\_mode\_t \* config )

Get reset configuration for registers related with Trigger Mode.

This function returns a configuration structure with reset values for features associated with Trigger Mode.

**Parameters**

<i>config</i>	- the configuration structure
---------------	-------------------------------

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 422 of file cmp\_driver.c.

**16.32.6.16** status\_t CMP\_DRV\_GetInputFlags ( const uint32\_t instance, cmp\_ch\_list\_t \* flags )

Gets input channels change flags.

This function return in <flags> all input channels flags as uint8\_t(1 bite for each channel flag).

**Parameters**

<i>instance</i>	- instance number
<i>flags</i>	- pointer to input flags

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 544 of file cmp\_driver.c.

**16.32.6.17** status\_t CMP\_DRV\_GetMUXConfig ( const uint32\_t instance, cmp\_anmux\_t \*const config )

Return configuration only for the MUX component.

This function returns current configuration to determine which signals go to comparator ports.

**Parameters**

<i>instance</i>	- instance number
<i>config</i>	- the configuration structure

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 402 of file cmp\_driver.c.

16.32.6.18 `status_t CMP_DRV_GetOutputFlags ( const uint32_t instance, cmp_output_trigger_t * flags )`

Get comparator output flags.

This function returns in <flags> comparator output flags(rising and falling edge).

**Parameters**

<i>instance</i>	- instance number
-	flags - pointer to output flags NO_EVENT RISING_EDGE FALLING_EDGE BOTH_EDGE

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 499 of file cmp\_driver.c.

16.32.6.19 `status_t CMP_DRV_GetTriggerModeConfig ( const uint32_t instance, cmp_trigger_mode_t *const config )`

Get current trigger mode configuration.

This function returns the current trigger mode configuration for trigger mode.

**Parameters**

<i>instance</i>	- instance number
<i>config</i>	- the configuration structure

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 472 of file cmp\_driver.c.

16.32.6.20 `status_t CMP_DRV_Init ( const uint32_t instance, const cmp_module_t *const config )`

Configure all comparator features with the given configuration structure.

This function configures the comparator module with the options provided in the config structure.

**Parameters**

<i>instance</i>	- instance number
<i>config</i>	- the configuration structure

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 188 of file cmp\_driver.c.

16.32.6.21 `status_t CMP_DRV_Reset ( const uint32_t instance )`

Reset all registers.

This function set all CMP registers to reset values.

**Parameters**

<i>instance</i>	- instance number
-----------------	-------------------

**Returns**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 66 of file cmp\_driver.c.

## 16.33 Controller Area Network - Peripheral Abstraction Layer (CAN PAL)

### 16.33.1 Detailed Description

The S32 SDK provides a Peripheral Abstraction Layer for Controller Area Network (CAN) modules of S32 SDK devices.

The CAN PAL driver allows communication over a CAN bus. It was designed to be portable across all platforms and IPs which support CAN communication.

#### How to integrate CAN PAL in your application

Unlike the other drivers, CAN PAL modules need to include a configuration file named `can_pal_cfg.h`, which allows the user to specify which IPs are used and how many resources are allocated for each of them (state structures). The following code example shows how to configure one instance for each available CAN IP.

```
#ifndef can_pal_cfg_H
#define can_pal_cfg_H

/* Define which IP instance will be used in current project */
#define CAN_OVER_FLEXCAN

/* Define the resources necessary for current project */
#define NO_OF_FLEXCAN_INSTS_FOR_CAN    1U

#endif /* can_pal_cfg_H */
```

The following table contains the matching between platforms and available IPs

IP↔ M↔ CU	S32↔ K116	S32↔ K118	S32↔ K142	S32↔ K144	S32↔ K146	S32↔ K148	S32↔ V234	MP↔ C5748 G	MP↔ C5746 C	MP↔ C5744 P	S32↔ R274	S32↔ R372
Flex↔ CAN	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES	YES

#### Features

- Standard data frames
- Extended data frames
- Flexible data rate (FD)
- Bitrate switch inside FD format frames (BRS)
- Zero to sixty four bytes data length
- Programmable bit rate
- Flexible buffers configurable to store 0 to 8, 16, 32 or 64 bytes data length depending of platform support
- Each buffer configurable as receive or transmit, all supporting standard and extended messages
- Individual Rx Masking per buffer
- Loop-Back mode
- Remote request frames

The following table contains the matching between platforms and available features

FE↔ A↔ T↔ U↔ R↔ E/↔ M↔ CU	S32↔ K116	S32↔ K118	S32↔ K142	S32↔ K144	S32↔ K146	S32↔ K148	S32↔ V234	M↔ P↔ C5748 G	M↔ P↔ C5746 C	M↔ P↔ C5744 P	S32↔ R274	S32↔ R372
F↔ D/↔ B↔ RS	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	NO	Y↔ ES	Y↔ ES
data length > 8B	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	NO	Y↔ ES	Y↔ ES

### Functionality

### Initialization

In order to use the CAN PAL driver it must be first initialized, using [CAN\\_Init\(\)](#) function. Once initialized, it cannot be initialized again for the same CAN module instance until it is de-initialized, using [CAN\\_Deinit\(\)](#). Different CAN modules instances can function independently of each other.

The [can\\_user\\_config\\_t](#) structure allows you to configure the following:

- the number of buffers needed;
- the operation mode, which can be one of the following:
  - normal mode;
  - loopback mode;
  - disable mode;
- the Protocol Engine clock source:
  - oscillator clock;
  - peripheral clock;
- the payload size of the buffers:
  - 8 bytes;
  - 16 bytes (only available with the FD feature enabled);
  - 32 bytes (only available with the FD feature enabled);
  - 64 bytes (only available with the FD feature enabled);
- enable/disable the Flexible Data-rate feature;
- the bitrate used for standard frames or for the arbitration phase of FD frames;
- the bitrate used for the data phase of FD frames;

The bitrate is represented by a [can\\_time\\_segment\\_t](#) structure, with the following fields:

- propagation segment;
- phase segment 1;
- phase segment 2;
- clock prescaler division factor;

- resync jump width.

In order to use a buffer for transmission/reception, it has to be initialized using either **CAN\_ConfigRxBuff** or **CAN\_ConfigTxBuff**.

After having the buffer configured, you can start sending/receiving data by calling one of the following functions:

- CAN\_Send;
- CAN\_SendBlocking;
- CAN\_Receive;
- CAN\_ReceiveBlocking.

#### FlexCAN Rx FIFO extension

When used over FlexCAN, the PAL allows extending the basic configuration with an Rx FIFO feature. The Rx FIFO is receive-only and 6-message deep. The application can read the received messages sequentially, in the order they were received, by repeatedly reading the data from buffer 0 (zero). A powerful filtering scheme is provided to accept only frames intended for the target application. The FIFO and filtering criteria are configured by passing a structure of **extension\_flexcan\_rx\_fifo\_t** type, through the extension field of the user configuration structure.

```
/* ID Filter table */
flexcan_id_table_t filterTable[] = {
    {
        .isExtendedFrame = false,
        .isRemoteFrame = false,
        .id = 1U
    },
    ...
};

/* Rx FIFO extension */
extension_flexcan_rx_fifo_t can_pall_rx_fifo_ext0 = {
    .numIdFilters = FLEXCAN_RX_FIFO_ID_FILTERS_8,
    .idFormat = FLEXCAN_RX_FIFO_ID_FORMAT_A,
    /* User must pass reference to the ID filter table. */
    .idFilterTable = NULL
};

can_pall_rx_fifo_ext0.idFilterTable = filterTable;
```

The number of elements in the ID filter table is defined by the following formula:

- for format A: the number of Rx FIFO ID filters
- for format B: twice the number of Rx FIFO ID filters
- for format C: four times the number of Rx FIFO ID filters The user must provide the exact number of elements in order to avoid any misconfiguration.

Each element in the ID filter table specifies an ID to be used as acceptance criteria for the FIFO as follows:

- for format A: In the standard frame format, bits 10 to 0 of the ID are used for frame identification. In the extended frame format, bits 28 to 0 are used.
- for format B: In the standard frame format, bits 10 to 0 of the ID are used for frame identification. In the extended frame format, only the 14 most significant bits (28 to 15) of the ID are compared to the 14 most significant bits (28 to 15) of the received ID.
- for format C: In both standard and extended frame formats, only the 8 most significant bits (7 to 0 for standard, 28 to 21 for extended) of the ID are compared to the 8 most significant bits (7 to 0 for standard, 28 to 21 for extended) of the received ID.

When Rx FIFO feature is enabled, buffer 0 (zero) cannot be used for transmission or reconfigured for reception using **CAN\_ConfigRxBuff()** and **CAN\_SetRxFilter()** functions.

## Important Notes

- Before using the CAN PAL driver the module clock must be configured. Refer to **Clock Manager** component for clock configuration.
- The driver enables the interrupts for the corresponding CAN module, but any interrupt priority must be done by the application.
- The board specific configurations must be done prior to driver calls; the driver has no influence on the functionality of the RX/TX pins - they must be configured by application. Refer to **PinSettings** component for pin configuration.
- Some features are not available for all platforms (see the table above for the matching between platforms and available features).
- When used **CAN\_ReceiveBlocking()** and **CAN\_SendBlocking()** with timeout parameter 0 and the message is already in mailbox configured will report timeout and successful transmit or receive the message.

## ## Example code ##

```
#define TX_BUFF_IDX      0U
#define RX_BUFF_IDX      1U

uint32_t msgID = 0xAB;

/* CAN PAL instance information */
const can_instance_t can_pall_instance = {CAN_INST_TYPE_FLEXCAN, 0U};

/* User configuration structure */
can_user_config_t config = {
    .maxBuffNum = 2U,
    .mode = CAN_LOOPBACK_MODE,
    .peClkSrc = CAN_CLK_SOURCE_OSC,
    .enableFD = false,
    .payloadSize = CAN_PAYLOAD_SIZE_8,
    .nominalBitrate = {
        .propSeg = 7,
        .phaseSeg1 = 4,
        .phaseSeg2 = 1,
        .preDivider = 0,
        .rJumpwidth = 1
    },
    .dataBitrate = {
        .propSeg = 7,
        .phaseSeg1 = 4,
        .phaseSeg2 = 1,
        .preDivider = 0,
        .rJumpwidth = 1
    },
    .extension = NULL
};

/* Initialize CAN */
CAN_Init(&can_pall_instance, &config);

/* Buffer configuration */
can_buff_config_t buffConfig = {
    .enableFD = false,
    .enableBRS = false,
    .fdPadding = 0xCC,
    .idType = CAN_MSG_ID_STD,
    .isRemote = false
};

CAN_ConfigTxBuff(&can_pall_instance, TX_BUFF_IDX, &buffConfig);
CAN_ConfigRxBuff(&can_pall_instance, RX_BUFF_IDX, &buffConfig, msgID);

can_message_t recvMsg, sendMsg = {
    .id = msgID,
    .length = 5U,
    .data = {"Hello"}
};

/* Send data using buffer configured for transmission */
CAN_Send(&can_pall_instance, TX_BUFF_IDX, &sendMsg);
while(CAN_GetTransferStatus(&can_pall_instance, TX_BUFF_IDX) == STATUS_BUSY);

/* Receive data using buffer configured for reception */
CAN_Receive(&can_pall_instance, RX_BUFF_IDX, &recvMsg);
while(CAN_GetTransferStatus(&can_pall_instance, RX_BUFF_IDX) == STATUS_BUSY);
```



```
/* De-initialize CAN */
CAN_Deinit(&can_pal_instance);
```

### Integration guideline

### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\pal\src\can\can_pal.c
```

### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\pal\inc\
```

### Preprocessor symbols

No special symbols are required for this component

### Dependencies

Controller Area Network with Flexible Data Rate (FlexCAN) Clock Manager Interrupt Manager (Interrupt) Enhanced Direct Memory Access (eDMA)

### Data Structures

- struct [can\\_time\\_segment\\_t](#)  
*CAN bit timing variables Implements : [can\\_time\\_segment\\_t\\_Class](#). [More...](#)*
- struct [can\\_buff\\_config\\_t](#)  
*CAN buffer configuration Implements : [can\\_buff\\_config\\_t\\_Class](#). [More...](#)*
- struct [can\\_message\\_t](#)  
*CAN message format Implements : [can\\_message\\_t\\_Class](#). [More...](#)*
- struct [can\\_user\\_config\\_t](#)  
*CAN controller configuration Implements : [can\\_user\\_config\\_t\\_Class](#). [More...](#)*
- struct [extension\\_flexcan\\_rx\\_fifo\\_t](#)  
*FlexCAN Rx FIFO configuration Implements : [extension\\_flexcan\\_rx\\_fifo\\_t\\_Class](#). [More...](#)*

### Enumerations

- enum [can\\_operation\\_modes\\_t](#) { [CAN\\_NORMAL\\_MODE](#) = 0U, [CAN\\_LOOPBACK\\_MODE](#) = 2U, [CAN\\_DISABLE\\_MODE](#) = 4U }
- CAN controller operation modes Implements : [can\\_operation\\_modes\\_t\\_Class](#).*
- enum [can\\_fd\\_payload\\_size\\_t](#) { [CAN\\_PAYLOAD\\_SIZE\\_8](#) = 0, [CAN\\_PAYLOAD\\_SIZE\\_16](#), [CAN\\_PAYLOAD\\_SIZE\\_32](#), [CAN\\_PAYLOAD\\_SIZE\\_64](#) }
- CAN buffer payload sizes Implements : [can\\_fd\\_payload\\_size\\_t\\_Class](#).*
- enum [can\\_bitrate\\_phase\\_t](#) { [CAN\\_NOMINAL\\_BITRATE](#), [CAN\\_FD\\_DATA\\_BITRATE](#) }
- CAN bitrate phase (nominal/data) Implements : [can\\_bitrate\\_phase\\_t\\_Class](#).*
- enum [can\\_msg\\_id\\_type\\_t](#) { [CAN\\_MSG\\_ID\\_STD](#), [CAN\\_MSG\\_ID\\_EXT](#) }
- CAN Message Buffer ID type Implements : [can\\_msg\\_id\\_type\\_t\\_Class](#).*
- enum [can\\_clk\\_source\\_t](#) { [CAN\\_CLK\\_SOURCE\\_OSC](#) = 0U, [CAN\\_CLK\\_SOURCE\\_PERIPH](#) = 1U }
- CAN PE clock sources Implements : [can\\_clk\\_source\\_t\\_Class](#).*

## Functions

- status\_t **CAN\_Init** (const [can\\_instance\\_t](#) \*const instance, const [can\\_user\\_config\\_t](#) \*config)  
*Initializes the CAN module.*
- status\_t **CAN\_Deinit** (const [can\\_instance\\_t](#) \*const instance)  
*De-initializes the CAN module.*
- status\_t **CAN\_SetBtrRate** (const [can\\_instance\\_t](#) \*const instance, [can\\_btr\\_rate\\_t](#) phase, const [can\\_time\\_segment\\_t](#) \*bitTiming)  
*Configures the CAN btrRate.*
- status\_t **CAN\_GetBtrRate** (const [can\\_instance\\_t](#) \*const instance, [can\\_btr\\_rate\\_t](#) phase, [can\\_time\\_segment\\_t](#) \*bitTiming)  
*Returns the CAN btrRate.*
- status\_t **CAN\_ConfigTxBuff** (const [can\\_instance\\_t](#) \*const instance, uint32\_t buffIdx, const [can\\_buff\\_config\\_t](#) \*config)  
*Configures a buffer for transmission.*
- status\_t **CAN\_ConfigRemoteResponseBuff** (const [can\\_instance\\_t](#) \*const instance, uint32\_t buffIdx, const [can\\_buff\\_config\\_t](#) \*config, const [can\\_message\\_t](#) \*message)  
*Configures a transmit buffer for remote frame response.*
- status\_t **CAN\_ConfigRxBuff** (const [can\\_instance\\_t](#) \*const instance, uint32\_t buffIdx, const [can\\_buff\\_config\\_t](#) \*config, uint32\_t acceptedId)  
*Configures a buffer for reception.*
- status\_t **CAN\_Send** (const [can\\_instance\\_t](#) \*const instance, uint32\_t buffIdx, const [can\\_message\\_t](#) \*message)  
*Sends a CAN frame using the specified buffer.*
- status\_t **CAN\_SendBlocking** (const [can\\_instance\\_t](#) \*const instance, uint32\_t buffIdx, const [can\\_message\\_t](#) \*message, uint32\_t timeoutMs)  
*Sends a CAN frame using the specified buffer, in a blocking manner.*
- status\_t **CAN\_Receive** (const [can\\_instance\\_t](#) \*const instance, uint32\_t buffIdx, [can\\_message\\_t](#) \*message)  
*Receives a CAN frame using the specified message buffer.*
- status\_t **CAN\_ReceiveBlocking** (const [can\\_instance\\_t](#) \*const instance, uint32\_t buffIdx, [can\\_message\\_t](#) \*message, uint32\_t timeoutMs)  
*Receives a CAN frame using the specified buffer, in a blocking manner.*
- status\_t **CAN\_AbortTransfer** (const [can\\_instance\\_t](#) \*const instance, uint32\_t buffIdx)  
*Ends a non-blocking CAN transfer early.*
- status\_t **CAN\_SetRxFilter** (const [can\\_instance\\_t](#) \*const instance, [can\\_msg\\_id\\_type\\_t](#) idType, uint32\_t buffIdx, uint32\_t mask)  
*Configures an ID filter for a specific reception buffer.*
- status\_t **CAN\_GetTransferStatus** (const [can\\_instance\\_t](#) \*const instance, uint32\_t buffIdx)  
*Returns the state of the previous CAN transfer.*
- status\_t **CAN\_InstallEventCallback** (const [can\\_instance\\_t](#) \*const instance, [can\\_callback\\_t](#) callback, void \*callbackParam)  
*Installs a callback function for the IRQ handler.*
- void **CAN\_GetDefaultConfig** ([can\\_instance\\_t](#) \*instance, [can\\_user\\_config\\_t](#) \*config)  
*Returns the Default configuration for CAN\_PAL instance 0 over FlexCan with a 500K Baud in normal mode, without flexible datarate with oscillator clock source for PE and 8 Bytes payload size.*

### 16.33.2 Data Structure Documentation

#### 16.33.2.1 struct [can\\_time\\_segment\\_t](#)

CAN bit timing variables Implements : [can\\_time\\_segment\\_t](#)\_Class.

Definition at line 59 of file [can\\_pal.h](#).

#### Data Fields

- uint32\_t [propSeg](#)
- uint32\_t [phaseSeg1](#)
- uint32\_t [phaseSeg2](#)
- uint32\_t [preDivider](#)
- uint32\_t [rJumpwidth](#)

#### Field Documentation

##### 16.33.2.1.1 uint32\_t phaseSeg1

Phase segment 1

Definition at line 61 of file can\_pal.h.

##### 16.33.2.1.2 uint32\_t phaseSeg2

Phase segment 2

Definition at line 62 of file can\_pal.h.

##### 16.33.2.1.3 uint32\_t preDivider

Clock prescaler division factor

Definition at line 63 of file can\_pal.h.

##### 16.33.2.1.4 uint32\_t propSeg

Propagation segment

Definition at line 60 of file can\_pal.h.

##### 16.33.2.1.5 uint32\_t rJumpwidth

Resync jump width

Definition at line 64 of file can\_pal.h.

##### 16.33.2.2 struct can\_buff\_config\_t

CAN buffer configuration Implements : can\_buff\_config\_t\_Class.

Definition at line 94 of file can\_pal.h.

#### Data Fields

- bool [enableFD](#)
- bool [enableBRS](#)
- uint8\_t [fdPadding](#)
- [can\\_msg\\_id\\_type\\_t](#) idType
- bool [isRemote](#)

#### Field Documentation

##### 16.33.2.2.1 bool enableBRS

Enable bit rate switch inside a CAN FD frame

Definition at line 96 of file can\_pal.h.

#### 16.33.2.2.2 `bool enableFD`

Enable flexible data rate

Definition at line 95 of file `can_pal.h`.

#### 16.33.2.2.3 `uint8_t fdPadding`

Value used for padding when the data length code (DLC) specifies a bigger payload size than the actual data length

Definition at line 97 of file `can_pal.h`.

#### 16.33.2.2.4 `can_msg_id_type_t idType`

Specifies whether the frame format is standard or extended

Definition at line 99 of file `can_pal.h`.

#### 16.33.2.2.5 `bool isRemote`

Specifies if the frame is standard or remote

Definition at line 100 of file `can_pal.h`.

#### 16.33.2.3 `struct can_message_t`

CAN message format Implements : `can_message_t_Class`.

Definition at line 106 of file `can_pal.h`.

##### Data Fields

- `uint32_t cs`
- `uint32_t id`
- `uint8_t data` [64]
- `uint8_t length`

##### Field Documentation

#### 16.33.2.3.1 `uint32_t cs`

Code and Status

Definition at line 107 of file `can_pal.h`.

#### 16.33.2.3.2 `uint8_t data`[64]

Data bytes of the CAN message

Definition at line 109 of file `can_pal.h`.

#### 16.33.2.3.3 `uint32_t id`

ID of the message

Definition at line 108 of file `can_pal.h`.

#### 16.33.2.3.4 `uint8_t length`

Length of payload in bytes

Definition at line 110 of file `can_pal.h`.

#### 16.33.2.4 `struct can_user_config_t`

CAN controller configuration Implements : `can_user_config_t_Class`.

Definition at line 116 of file can\_pal.h.

#### Data Fields

- uint32\_t [maxBuffNum](#)
- [can\\_operation\\_modes\\_t](#) mode
- [can\\_clk\\_source\\_t](#) peClkSrc
- bool [enableFD](#)
- [can\\_fd\\_payload\\_size\\_t](#) payloadSize
- [can\\_time\\_segment\\_t](#) nominalBitrate
- [can\\_time\\_segment\\_t](#) dataBitrate
- void \* [extension](#)

#### Field Documentation

##### 16.33.2.4.1 [can\\_time\\_segment\\_t](#) dataBitrate

Bit timing segments for data bitrate

Definition at line 124 of file can\_pal.h.

##### 16.33.2.4.2 [bool](#) enableFD

Enable flexible data rate

Definition at line 121 of file can\_pal.h.

##### 16.33.2.4.3 [void\\*](#) extension

This field will be used to add extra settings to the basic configuration like FlexCAN Rx FIFO settings

Definition at line 125 of file can\_pal.h.

##### 16.33.2.4.4 [uint32\\_t](#) maxBuffNum

Set maximum number of buffers

Definition at line 118 of file can\_pal.h.

##### 16.33.2.4.5 [can\\_operation\\_modes\\_t](#) mode

Set operation mode

Definition at line 119 of file can\_pal.h.

##### 16.33.2.4.6 [can\\_time\\_segment\\_t](#) nominalBitrate

Bit timing segments for nominal bitrate

Definition at line 123 of file can\_pal.h.

##### 16.33.2.4.7 [can\\_fd\\_payload\\_size\\_t](#) payloadSize

Set size of buffer payload

Definition at line 122 of file can\_pal.h.

##### 16.33.2.4.8 [can\\_clk\\_source\\_t](#) peClkSrc

The clock source of the CAN Protocol Engine (PE).

Definition at line 120 of file can\_pal.h.

### 16.33.2.5 struct extension\_flexcan\_rx\_fifo\_t

FlexCAN Rx FIFO configuration Implements : extension\_flexcan\_rx\_fifo\_t\_Class.

Definition at line 133 of file can\_pal.h.

#### Data Fields

- flexcan\_rx\_fifo\_id\_filter\_num\_t numIdFilters
- flexcan\_rx\_fifo\_id\_element\_format\_t idFormat
- flexcan\_id\_table\_t \* idFilterTable

#### Field Documentation

#### 16.33.2.5.1 flexcan\_id\_table\_t\* idFilterTable

Rx FIFO ID table

Definition at line 137 of file can\_pal.h.

#### 16.33.2.5.2 flexcan\_rx\_fifo\_id\_element\_format\_t idFormat

RX FIFO ID format

Definition at line 136 of file can\_pal.h.

#### 16.33.2.5.3 flexcan\_rx\_fifo\_id\_filter\_num\_t numIdFilters

The number of Rx FIFO ID filters needed

Definition at line 135 of file can\_pal.h.

### 16.33.3 Enumeration Type Documentation

#### 16.33.3.1 enum can\_bitrate\_phase\_t

CAN bitrate phase (nominal/data) Implements : can\_bitrate\_phase\_t\_Class.

#### Enumerator

**CAN\_NOMINAL\_BITRATE** Nominal (FD arbitration) bitrate

**CAN\_FD\_DATA\_BITRATE** FD data bitrate

Definition at line 70 of file can\_pal.h.

#### 16.33.3.2 enum can\_clk\_source\_t

CAN PE clock sources Implements : can\_clk\_source\_t\_Class.

#### Enumerator

**CAN\_CLK\_SOURCE\_OSC** The CAN engine clock source is the oscillator clock.

**CAN\_CLK\_SOURCE\_PERIPH** The CAN engine clock source is the peripheral clock.

Definition at line 86 of file can\_pal.h.

#### 16.33.3.3 enum can\_fd\_payload\_size\_t

CAN buffer payload sizes Implements : can\_fd\_payload\_size\_t\_Class.

#### Enumerator

**CAN\_PAYLOAD\_SIZE\_8** CAN message buffer payload size in bytes

**CAN\_PAYLOAD\_SIZE\_16** CAN message buffer payload size in bytes

**CAN\_PAYLOAD\_SIZE\_32** CAN message buffer payload size in bytes

**CAN\_PAYLOAD\_SIZE\_64** CAN message buffer payload size in bytes

Definition at line 49 of file can\_pal.h.

#### 16.33.3.4 enum can\_msg\_id\_type\_t

CAN Message Buffer ID type Implements : can\_msg\_id\_type\_t\_Class.

##### Enumerator

**CAN\_MSG\_ID\_STD** Standard ID

**CAN\_MSG\_ID\_EXT** Extended ID

Definition at line 78 of file can\_pal.h.

#### 16.33.3.5 enum can\_operation\_modes\_t

CAN controller operation modes Implements : can\_operation\_modes\_t\_Class.

##### Enumerator

**CAN\_NORMAL\_MODE** Normal mode or user mode

**CAN\_LOOPBACK\_MODE** Loop-back mode

**CAN\_DISABLE\_MODE** Module disable mode

Definition at line 40 of file can\_pal.h.

### 16.33.4 Function Documentation

#### 16.33.4.1 status\_t CAN\_AbortTransfer ( const can\_instance\_t \*const instance, uint32\_t buffidx )

Ends a non-blocking CAN transfer early.

##### Note

When the Rx FIFO extension is used, buffer 0 (zero) is used to read the contents of the FIFO and is configured at the initialization of the driver.

##### Parameters

in	<i>instance</i>	Instance information structure.
in	<i>buffidx</i>	buffer index.

##### Returns

STATUS\_SUCCESS if successful; STATUS\_CAN\_NO\_TRANSFER\_IN\_PROGRESS if no transfer was running

Definition at line 903 of file can\_pal.c.

#### 16.33.4.2 status\_t CAN\_ConfigRemoteResponseBuff ( const can\_instance\_t \*const instance, uint32\_t buffidx, const can\_buff\_config\_t \* config, const can\_message\_t \* message )

Configures a transmit buffer for remote frame response.

**Parameters**

in	<i>instance</i>	Instance information structure.
in	<i>buffidx</i>	buffer index.
in	<i>config</i>	buffer configuration.
in	<i>message</i>	frame to be sent as remote response.

**Returns**

STATUS\_SUCCESS if successful; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the buffer index is out of range; STATUS\_ERROR if invalid instance number is used;

Definition at line 553 of file can\_pal.c.

16.33.4.3 **status\_t** CAN\_ConfigRxBuff ( **const** can\_instance\_t \*const *instance*, uint32\_t *buffidx*, **const** can\_buff\_config\_t \* *config*, uint32\_t *acceptedId* )

Configures a buffer for reception.

This function configures a buffer for reception.

**Note**

When the Rx FIFO extension is used, buffer 0 (zero) is used to read the contents of the FIFO and is configured at the initialization of the driver. The user should not reconfigure this buffer for classical buffer reception.

**Parameters**

in	<i>instance</i>	Instance information structure.
in	<i>buffidx</i>	buffer index.
in	<i>config</i>	buffer configuration.
in	<i>acceptedId</i>	ID used for accepting frames.

**Returns**

STATUS\_SUCCESS if successful; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the buffer index is out of range; STATUS\_ERROR if invalid instance number is used;

Definition at line 616 of file can\_pal.c.

16.33.4.4 **status\_t** CAN\_ConfigTxBuff ( **const** can\_instance\_t \*const *instance*, uint32\_t *buffidx*, **const** can\_buff\_config\_t \* *config* )

Configures a buffer for transmission.

This function configures a buffer for transmission.

**Note**

When the Rx FIFO extension is used, buffer 0 (zero) is used to read the contents of the FIFO and is configured at the initialization of the driver. The user should not reconfigure this buffer for transmission.

**Parameters**

in	<i>instance</i>	Instance information structure.
in	<i>buffidx</i>	buffer index.



<i>in</i>	<i>config</i>	buffer configuration.
-----------	---------------	-----------------------

**Returns**

STATUS\_SUCCESS if successful; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the buffer index is out of range; STATUS\_ERROR if invalid instance number is used;

Definition at line 489 of file can\_pal.c.

#### 16.33.4.5 status\_t CAN\_Deinit ( const can\_instance\_t \*const *instance* )

De-initializes the CAN module.

This function de-initializes the CAN module.

**Parameters**

<i>in</i>	<i>instance</i>	Instance information structure
-----------	-----------------	--------------------------------

**Returns**

STATUS\_SUCCESS if successful; STATUS\_ERROR if unsuccessful or invalid instance number;

Definition at line 369 of file can\_pal.c.

#### 16.33.4.6 status\_t CAN\_GetBitrate ( const can\_instance\_t \*const *instance*, can\_bitrate\_phase\_t *phase*, can\_time\_segment\_t \* *bitTiming* )

Returns the CAN bitrate.

This function returns the CAN configured bitrate.

**Parameters**

<i>in</i>	<i>instance</i>	Instance information structure.
<i>in</i>	<i>phase</i>	selects between nominal/data phase bitrate.
<i>out</i>	<i>bitTiming</i>	configured bit timing variables.

**Returns**

STATUS\_SUCCESS if successful; STATUS\_ERROR if invalid instance number is used;

Definition at line 449 of file can\_pal.c.

#### 16.33.4.7 void CAN\_GetDefaultConfig ( can\_instance\_t \* *instance*, can\_user\_config\_t \* *config* )

Returns the Default configuration for CAN\_PAL instance 0 over FlexCan with a 500K Baud in normal mode, without flexible datarate with oscillator clock source for PE and 8 Bytes payload size.

**Parameters**

<i>out</i>	<i>instance</i>	Pointer for can_instance structure.
<i>out</i>	<i>config</i>	Pointer for can_user_config structure.

Definition at line 1072 of file can\_pal.c.

#### 16.33.4.8 status\_t CAN\_GetTransferStatus ( const can\_instance\_t \*const *instance*, uint32\_t *buffIdx* )

Returns the state of the previous CAN transfer.

When performing an async transfer, call this function to ascertain the state of the current transfer: in progress or complete.

**Note**

When the Rx FIFO extension is used, buffer 0 (zero) is used to read the contents of the FIFO and is configured at the initialization of the driver.

**Parameters**

in	<i>instance</i>	Instance information structure.
in	<i>buffidx</i>	buffer index.

**Returns**

STATUS\_SUCCESS if successful; STATUS\_BUSY if a resource is busy; STATUS\_ERROR if invalid instance number is used;

Definition at line 985 of file can\_pal.c.

**16.33.4.9** `status_t CAN_Init ( const can_instance_t *const instance, const can_user_config_t * config )`

Initializes the CAN module.

This function initializes and enables the requested CAN module.

**Note**

When the Rx FIFO extension is used, buffer 0 (zero) is used to read the contents of the FIFO and is configured at the initialization of the driver.

**Parameters**

in	<i>instance</i>	Instance information structure
in	<i>config</i>	The configuration structure

**Returns**

STATUS\_SUCCESS if successful; STATUS\_ERROR if unsuccessful or invalid instance number;

Definition at line 263 of file can\_pal.c.

**16.33.4.10** `status_t CAN_InstallEventCallback ( const can_instance_t *const instance, can_callback_t callback, void * callbackParam )`

Installs a callback function for the IRQ handler.

**Parameters**

in	<i>instance</i>	Instance information structure.
in	<i>callback</i>	The callback function.
in	<i>callbackParam</i>	User parameter passed to the callback function through the state parameter.

**Returns**

STATUS\_SUCCESS if successful; STATUS\_ERROR if invalid instance number is used;

Definition at line 1024 of file can\_pal.c.

**16.33.4.11** `status_t CAN_Receive ( const can_instance_t *const instance, uint32_t buffidx, can_message_t * message )`

Receives a CAN frame using the specified message buffer.

This function receives a CAN frame using a configured buffer. The function returns immediately. If a callback is installed, it will be invoked after the frame was received and read into the specified buffer.

**Note**

When the Rx FIFO extension is used, buffer 0 (zero) is used to read the contents of the FIFO and is configured at the initialization of the driver. The user should use this buffer to receive frames in the FIFO.

**Parameters**

in	<i>instance</i>	Instance information structure.
in	<i>buffidx</i>	buffer index.
out	<i>message</i>	received message.

**Returns**

STATUS\_SUCCESS if successful; STATUS\_BUSY if the current buffer is involved in another transfer; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the buffer index is out of range; STATUS\_ERROR if invalid instance number is used;

Definition at line 797 of file can\_pal.c.

**16.33.4.12** `status_t CAN_ReceiveBlocking ( const can_instance_t *const instance, uint32_t buffidx, can_message_t *message, uint32_t timeoutMs )`

Receives a CAN frame using the specified buffer, in a blocking manner.

This function receives a CAN frame using a configured buffer. The function blocks until either a frame was received, or the specified timeout expired.

**Note**

When the Rx FIFO extension is used, buffer 0 (zero) is used to read the contents of the FIFO and is configured at the initialization of the driver. The user should use this buffer to receive frames in the FIFO.

**Parameters**

in	<i>instance</i>	Instance information structure.
in	<i>buffidx</i>	buffer index.
out	<i>message</i>	received message.
in	<i>timeoutMs</i>	A timeout for the transfer in milliseconds.

**Returns**

STATUS\_SUCCESS if successful; STATUS\_BUSY if the current buffer is involved in another transfer; STATUS\_TIMEOUT if the timeout is reached; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the buffer index is out of range; STATUS\_ERROR if invalid instance number is used;

Definition at line 849 of file can\_pal.c.

**16.33.4.13** `status_t CAN_Send ( const can_instance_t *const instance, uint32_t buffidx, const can_message_t *message )`

Sends a CAN frame using the specified buffer.

This function sends a CAN frame using a configured buffer. The function returns immediately. If a callback is installed, it will be invoked after the frame was sent.

**Note**

When the Rx FIFO extension is used, buffer 0 (zero) is used to read the contents of the FIFO and is configured at the initialization of the driver. The user should not use this buffer for transmission.

**Parameters**

in	<i>instance</i>	Instance information structure.
in	<i>buffIdx</i>	buffer index.
in	<i>message</i>	message to be sent.

**Returns**

STATUS\_SUCCESS if successful; STATUS\_BUSY if the current buffer is involved in another transfer; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the buffer index is out of range; STATUS\_ERROR if invalid instance number is used;

Definition at line 678 of file can\_pal.c.

**16.33.4.14** `status_t CAN_SendBlocking ( const can_instance_t *const instance, uint32_t buffIdx, const can_message_t * message, uint32_t timeoutMs )`

Sends a CAN frame using the specified buffer, in a blocking manner.

This function sends a CAN frame using a configured buffer. The function blocks until either the frame was sent, or the specified timeoutMs expired.

**Note**

When the Rx FIFO extension is used, buffer 0 (zero) is used to read the contents of the FIFO and is configured at the initialization of the driver. The user should not use this buffer for transmission.

**Parameters**

in	<i>instance</i>	Instance information structure.
in	<i>buffIdx</i>	buffer index.
in	<i>message</i>	message to be sent.
in	<i>timeoutMs</i>	A timeout for the transfer in milliseconds.

**Returns**

STATUS\_SUCCESS if successful; STATUS\_BUSY if the current buffer is involved in another transfer; STATUS\_TIMEOUT if the timeout is reached; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the buffer index is out of range; STATUS\_ERROR if invalid instance number is used;

Definition at line 737 of file can\_pal.c.

**16.33.4.15** `status_t CAN_SetBtrRate ( const can_instance_t *const instance, can_bitrate_phase_t phase, const can_time_segment_t * bitTiming )`

Configures the CAN bitrate.

This function configures the CAN bit timing variables.

**Parameters**

in	<i>instance</i>	Instance information structure.
in	<i>phase</i>	selects between nominal/data phase bitrate.
in	<i>bitTiming</i>	bit timing variables.

**Returns**

STATUS\_SUCCESS if successful; STATUS\_ERROR if invalid instance number is used;

Definition at line 402 of file can\_pal.c.

16.33.4.16 `status_t CAN_SetRxFilter ( const can_instance_t *const instance, can_msg_id_type_t idType, uint32_t buffIdx, uint32_t mask )`

Configures an ID filter for a specific reception buffer.

This function configures an ID filter for each reception buffer.

#### Note

When the Rx FIFO extension is used, buffer 0 (zero) is used to read the contents of the FIFO and is configured at the initialization of the driver. The user should not reconfigure the Rx filter for this buffer.

#### Parameters

<code>in</code>	<code>instance</code>	Instance information structure.
<code>in</code>	<code>idType</code>	selects between standard and extended ID.
<code>in</code>	<code>buffIdx</code>	buffer index.
<code>in</code>	<code>mask</code>	mask value for ID filtering.

#### Returns

STATUS\_SUCCESS if successful; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the buffer index is out of range; STATUS\_ERROR if invalid instance number is used;

Definition at line 942 of file can\_pal.c.

## 16.34 Controller Area Network with Flexible Data Rate (FlexCAN)

### 16.34.1 Detailed Description

The S32 SDK provides a Peripheral Driver for the FlexCAN module of S32 SDK devices.

#### Hardware background

The FlexCAN module is a communication controller implementing the CAN protocol according to the ISO 11898-1 standard and CAN 2.0 B protocol specifications. The FlexCAN module is a full implementation of the CAN protocol specification, the CAN with Flexible Data rate (CAN FD) protocol and the CAN 2.0 version B protocol, which supports both standard and extended message frames and long payloads up to 64 bytes transferred at faster rates up to 8 Mbps. The message buffers are stored in an embedded RAM dedicated to the FlexCAN module.

The FlexCAN module includes these distinctive features:

- Full implementation of the CAN with Flexible Data Rate (CAN FD) protocol specification and CAN protocol specification, Version 2.0 B (see the `FEATURE_CAN_HAS_FD` define for the availability of this feature on each platform)
  - Standard data frames
  - Extended data frames
  - Zero to sixty four bytes data length
  - Programmable bit rate (see the chip-specific FlexCAN information for the specific maximum bit rate configuration)
  - Content-related addressing
- Compliant with the ISO 11898-1 standard
- Flexible mailboxes configurable to store 0 to 8, 16, 32 or 64 bytes data length (payloads longer than 8 bytes are available only for some platforms, see the `FEATURE_CAN_HAS_FD` define)
- Each mailbox configurable as receive or transmit, all supporting standard and extended messages
- Individual Rx Mask registers per mailbox
- Full-featured Rx FIFO with storage capacity for up to six frames and automatic internal pointer handling with DMA support (DMA support is available only for some platforms, see the `FEATURE_CAN_HAS_DMA_ENABLE` define)
- Transmission abort capability
- Flexible message buffers (MBs) configurable as Rx or Tx (see the `FEATURE_CAN_MAX_MB_NUM` define for the specific maximum number of message buffers configurable on each platform)
- Programmable clock source to the CAN Protocol Interface, either peripheral clock or oscillator clock (this feature might differ depending on the platform, see `FEATURE_CAN_HAS_PE_CLKSRC_SELECT` define for the availability of this feature on each platform)
- RAM not used by reception or transmission structures can be used as general purpose RAM space
- Listen-Only mode capability
- Programmable Loop-Back mode supporting self-test operation
- Maskable interrupts
- Short latency time due to an arbitration scheme for high-priority messages
- Low power modes or matching with received frames - Pretended Networking (see `FEATURE_CAN_HAS_PRETENDED_NETWORKING` define for the availability of this feature on each platform)
- Transceiver Delay Compensation feature when transmitting CAN FD messages at faster data rates (see the `FEATURE_CAN_HAS_FD` define for the availability of this feature on each platform)

- Remote request frames may be handled automatically or by software
- CAN bit time settings and configuration bits can only be written in Freeze mode
- SYNCH bit available in Error in Status 1 register to inform that the module is synchronous with CAN bus
- CRC status for transmitted message
- Rx FIFO Global Mask register
- Selectable priority between mailboxes and Rx FIFO during matching process
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 128 extended, 256 standard, or 512 partial (8 bit) IDs, with up to 32 individual masking capability
- 100% backward compatibility with previous FlexCAN version
- Supports Pretended Networking functionality in low power: Stop mode (see FEATURE\_CAN\_HAS\_PRETENDED\_NETWORKING define for the availability of this feature on each platform)
- Supports detection and correction of errors in memory read accesses. Errors in one bit can be corrected and errors in 2 bits can be detected but not corrected (this feature might not be available on some platforms, see chip-specific FlexCAN information for details)
- Supports Self Wake Up feature when FlexCAN is in a low power mode: Stop mode (see FEATURE\_CAN\_HAS\_SELF\_WAKE\_UP define for the availability of this feature on each platform)
- Disable Detection and Correction of Memory Errors Feature for devices that supports it. This feature can cause Freeze Mode of CAN interface. (see FEATURE\_CAN\_HAS\_MEM\_ERR\_DET define availability of the feature in module)

#### Modules

- [FlexCAN Driver](#)

## 16.35 Cooked API

### 16.35.1 Detailed Description

Cooked processing of diagnostic messages manages one complete message at a time.

#### Functions

- void [ld\\_send\\_message](#) (I\_ifc\_handle iii, I\_u16 length, I\_u8 NAD, const I\_u8 \*const data)  
*Pack the information specified by data and length into one or multiple diagnostic frames.*
- void [ld\\_receive\\_message](#) (I\_ifc\_handle iii, I\_u16 \*const length, I\_u8 \*const NAD, I\_u8 \*const data)  
*Prepare the LIN diagnostic module to receive one message and store it in the buffer pointed to by data.*
- I\_u8 [ld\\_tx\\_status](#) (I\_ifc\_handle iii)  
*Get the status of the last made call to ld\_send\_message.*
- I\_u8 [ld\\_rx\\_status](#) (I\_ifc\_handle iii)  
*Get the status of the last made call to ld\_send\_message.*

### 16.35.2 Function Documentation

#### 16.35.2.1 void ld\_receive\_message ( I\_ifc\_handle iii, I\_u16 \*const length, I\_u8 \*const NAD, I\_u8 \*const data )

Prepare the LIN diagnostic module to receive one message and store it in the buffer pointed to by data.

##### Parameters

in	iii	Lin interface handle
in	length	Length of data to receive
in	NAD	Node address of slave node
in	data	Data to be sent

##### Returns

void

Prepare the LIN diagnostic module to receive one message and store it in the buffer pointed to by data. At the call, length shall specify the maximum length allowed. When the reception has completed, length is changed to the actual length and NAD to the NAD in the message.

Definition at line 261 of file lin\_commontl\_api.c.

#### 16.35.2.2 I\_u8 ld\_rx\_status ( I\_ifc\_handle iii )

Get the status of the last made call to ld\_send\_message.

##### Parameters

in	iii	Lin interface handle
----	-----	----------------------

##### Returns

I\_u8

The call returns the status of the last made call to ld\_receive\_message. < br / > The following values can be returned: < br / > LD\_IN\_PROGRESS: The reception is not yet completed. < br / > LD\_COMPLETED: The reception has completed successfully and all < br / > information (length, NAD, data) is available. (You can < br / > also issue a new ld\_receive\_message call). This < br / > value is also returned after initialization of the < br / > transport layer. < br / > LD\_FAILED: The reception ended in an error. The data was only < br / > partially received and should not be trusted. Initialize < br / > before processing further transport layer messages. < br /



> For LIN2.0 and J2602 Users can make a new call to `ld_receive_message`. For LIN2.1 and above, the transport layer shall be reinitialized before processing further messages. To find out why a transmission has failed, check the status management function `l_ifc_read_status`. \* LD\_N\_CR\_TIMEOUT The reception failed because of a N\_Cr timeout (For LIN2.1 and above only) < br / > LD\_WRONG\_SN The reception failed because of an unexpected sequence number. (For LIN2.1 and above only)

Definition at line 307 of file `lin_commontl_api.c`.

**16.35.2.3** `void ld_send_message ( l_ifc_handle iii, l_u16 length, l_u8 NAD, const l_u8 *const data )`

Pack the information specified by data and length into one or multiple diagnostic frames.

#### Parameters

in	<i>iii</i>	Lin interface handle
in	<i>length</i>	Length of data to send
in	<i>NAD</i>	Node address of slave node
in	<i>data</i>	Data to be sent

#### Returns

void

Pack the information specified by data and length into one or multiple diagnostic frames. If the call is made in a master node application the frames are transmitted to the slave node with the address NAD. If the call is made in a slave node application the frames are transmitted to the master node with the address NAD. The parameter NAD is not used in slave nodes.

Definition at line 207 of file `lin_commontl_api.c`.

**16.35.2.4** `l_u8 ld_tx_status ( l_ifc_handle iii )`

Get the status of the last made call to `ld_send_message`.

#### Parameters

in	<i>iii</i>	Lin interface handle
----	------------	----------------------

#### Returns

l\_u8

Get the status of the last made call to `ld_send_message`. The following values can be returned: LD\_IN\_PROGRESS: The transmission is not yet completed. LD\_COMPLETED: The transmission has completed successfully (and you can issue a new `ld_send_message` call). This value is also returned after initialization of the transport layer. LD\_FAILED: The transmission ended in an error. The data was only partially sent. The transport layer shall be reinitialized before processing further messages. To find out why a transmission has failed, check the status management function `l_read_status`. For LIN2.0 and J2602 Users can make a new call to `ld_send_message`. For LIN2.1 and above, the transport layer shall be reinitialized before processing further messages. LD\_N\_AS\_TIMEOUT: The transmission failed because of a N\_As timeout. This applies for LIN2.1 and above only.

Definition at line 291 of file `lin_commontl_api.c`.

## 16.36 Cryptographic Services Engine (CSEc)

### 16.36.1 Detailed Description

The S32 SDK provides a Peripheral Driver for the Cryptographic Services Engine (CSEc) module of S32 SDK devices.

The FTFC module has added features to comply with the SHE specification. By using an embedded processor, firmware and hardware assisted AES-128 sub-block, the FTFC macro enables encryption, decryption and message generation and authentication algorithms for secure messaging applications. Additionally a TRNG and Miyaguchi-Prenell compression sub-blocks enables true random number generation (entropy generator for PRNG in AES sub-block).

#### Hardware background

Features of the CSEc module include:

- Secure cryptographic key storage (ranging from 3 to 21 user keys)
- AES-128 encryption and decryption
- AES-128 CMAC (Cipher-based Message Authentication Code) calculation and authentication
- ECB (Electronic Cypher Book) Mode - encryption and decryption
- CBC (Cipher Block Chaining) Mode - encryption and decryption
- True and Pseudo random number generation
- Miyaguchi-Prenell compression function
- Secure Boot Mode (user configurable)
  - Sequential Boot Mode
  - Parallel Boot Mode
  - Strict Sequential Boot Mode (unchangeable once set)

#### Modules

- [CSEc Driver](#)  
*Cryptographic Services Engine Peripheral Driver.*

## 16.37 Cyclic Redundancy Check (CRC)

### 16.37.1 Detailed Description

The S32 SDK provides Peripheral Drivers for the Cyclic Redundancy Check (CRC) module.

#### 1. CRC with S32K1xx and **S32K1xxW**:

- Generate 16/32-bit CRC code for error detection.
- Provides a programmable polynomial, seed, and other parameters required to implement 16/32-bit CRC standard.
- Calculate 16/32-bit code for 32 bits of data at a time.

#### 2. CRC with MPC574x and S32Rx7x:

- Generate 8/16/32-bit CRC code for error detection.
- Provides a programmable polynomial, seed, and other parameters required to implement 8/16/32-bit CRC standard.
- Calculate 8/16/32-bit code for 32 bits of data at a time.

**Important note when use CRC module with MPC574x and S32Rx7x devices:**

- When generating CRC-32 for the ITU-T V.42 standard the user needs to set SWAP\_BYTEWISE together with INV and SWAP.
- When generating CRC-16-CCITT(0x1021) standard the user needs to set SWAP\_BITWISE bit.

### Basic Operations of CRC

1. To initialize the CRC module, call [CRC\\_DRV\\_Init\(\)](#) function and pass the user configuration data structure to it. This is example code to configure the CRC driver using [CRC\\_DRV\\_GetDefaultConfig\(\)](#) function:

```
#define INST_CRC1 (0U)

/* Configuration structure crc1_InitConfig0 */
crc_user_config_t crc1_InitConfig0;

/* Get default configuration for CRC module: CRC-16-CCITT (0x1021) standard */
CRC_DRV_GetDefaultConfig(&crc1_InitConfig0);

/* Initializes the CRC */
CRC_DRV_Init(INST_CRC1, &crc1_InitConfig0);
```

2. To configure and operate the CRC module: Function [CRC\\_DRV\\_Configure\(\)](#) shall be used to write user configuration to CRC hardware module before starting operation by calling [CRC\\_DRV\\_WriteData\(\)](#). Finally, using [CRC\\_DRV\\_GetCrcResult\(\)](#) function to get the result of CRC calculation. This is example code to configure and get CRC block for S32K1xx:

```
#define INST_CRC1 (0U)

uint8_t buffer[] = { 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x30 };
uint32_t result;

/* Set the CRC configuration: CRC-16-CCITT (0x1021) standard */
CRC_DRV_Configure(INST_CRC1, &crc1_InitConfig0);
/* Write data to the current CRC calculation */
CRC_DRV_WriteData(INST_CRC1, buffer, 10U);
/* Get result of CRC calculation (0x3218U) */
result = CRC_DRV_GetCrcResult(INST_CRC1);

/* De-init */
CRC_DRV_Deinit(INST_CRC1);
```

3. To get result of 32-bit data call [CRC\\_DRV\\_GetCrc32\(\)](#) function.

```
#define INST_CRC1 (0U)

uint32_t seed = 0xFFFFU;
uint32_t data = 0x12345678U;
uint32_t result;

/* Get result of 32-bit data (0x30EC) at CRC-16-CCITT (0x1021) standard configuration mode */
result = CRC_DRV_GetCrc32(INST_CRC1, data, true, seed);
```

4. To get result of 16-bit data call [CRC\\_DRV\\_GetCrc16\(\)](#) function.

```
#define INST_CRC1 (0U)

uint32_t seed = 0xFFFFU;
uint16_t data = 0x1234U;
uint32_t result;

/* Get result of 16-bit data (0x0EC9) at CRC-16-CCITT (0x1021) standard configuration mode */
result = CRC_DRV_GetCrc16(INST_CRC1, data, true, seed);
```

5. To get current configuration of the CRC module, just call [CRC\\_DRV\\_GetConfig\(\)](#) function.

```
#define INST_CRC1 (0U)
crc_user_config_t crc1_InitConfig0;

/* Get current configuration of the CRC module */
CRC_DRV_GetConfig(INST_CRC1, &crc1_InitConfig0);
```

6. To Get default configuration of the CRC module, just call [CRC\\_DRV\\_GetDefaultConfig\(\)](#) function.

```
#define INST_CRC1 (0U)
crc_user_config_t crc1_InitConfig0;

/* Get default configuration of the CRC module */
CRC_DRV_GetDefaultConfig(&crc1_InitConfig0);
```

## Integration guideline

### Compilation units

The following files need to be compiled in the project: For **S32K1xx** and **S32K1xxW**:

```
${S32SDK_PATH}\platform\drivers\src\crc\crc_driver.c
${S32SDK_PATH}\platform\drivers\src\crc\crc_hw_access.c
```

For **MPC574x** and **S32Rx7x**:

```
${S32SDK_PATH}\platform\drivers\src\crc\crc_driver.c
${S32SDK_PATH}\platform\drivers\src\crc\crc_c55_hw_access.c
```

### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc\
```

### Compile symbols

No special symbols are required for this component

### Dependencies

### [Clock Manager](#)

### Modules

- [CRC Driver](#)

*Cyclic Redundancy Check Peripheral Driver.*

## 16.38 Diagnostic services

### 16.38.1 Detailed Description

Diagnostic services defines methods to implement diagnostic data transfer between a master node connected with a diagnostic tester and the slave nodes.

Three different classes of diagnostic nodes are supported.

The master node and the diagnostic tester are connected via a back-bone bus (e.g. CAN). The master node shall receive all diagnostic requests addressed to the slave nodes from the back-bone bus, and gateway them to the correct LIN cluster(s). Responses from the slave nodes shall be gatewayed back to the back-bone bus through the master node.

All diagnostic requests and responses (services) addressed to the slave nodes can be routed in the network layer (i.e. no application layer routing). In this case, the master node must implement the LIN transport protocol, see Transport Layer Specification, as well as the transport protocols used on the back-bone busses (e.g. ISO15765-2 on CAN).

Currently, LinStack support some service. With other service which LinStack doesn't support or user want to add action when any service is received, user can choose or create service in supported services of PEX GUI and use API of transport layer to implement it. in application.

Example in slave node:

```
for(;;)
{
    /* length shall specify the maximum length allowed */
    length = 106;
    ld_receive_message(LI0,&length, &nad, req_data);
    /* if receive READ_DATA_BY_IDENTIFIER master request successfully */
    if(diag_get_flag(LI0, LI0_DIAGSRV_READ_DATA_BY_IDENTIFIER_ORDER))
    {
        diag_clear_flag(LI0, LI0_DIAGSRV_READ_DATA_BY_IDENTIFIER_ORDER);
        /* implement what you want to do when receive this message
           length will return real length of this message
           req_data will contain SID and data of this message */
        /* send back response data */
        ld_send_message(LI0,17,nad, res_data);
    }
}
```

### Modules

- [Node configuration](#)  
*This group contains APIs that used for node configuration purpose.*
- [Node identification](#)  
*This group contains API that used for node identification purpose.*

### Functions

- void [diag\\_read\\_data\\_by\\_identifier](#) (l\_ifc\_handle iii, const l\_u8 NAD, const l\_u8 number\_of\_id, const l\_u16 \*const list\_of\_id)  
*This function reads data by identifier, Diagnostic Class II service (0x22).*
- void [diag\\_write\\_data\\_by\\_identifier](#) (l\_ifc\_handle iii, const l\_u8 NAD, l\_u16 data\_length, const l\_u8 \*const data)  
*Write Data by Identifier for a specified node - Diagnostic Class II service (0x2E)*
- void [diag\\_session\\_control](#) (l\_ifc\_handle iii, const l\_u8 NAD, const l\_u8 session\_type)  
*This function is used for master node only. It will pack data and send request to slave node with service ID = 0x10: Session control.*
- void [diag\\_fault\\_memory\\_read](#) (l\_ifc\_handle iii, const l\_u8 NAD, l\_u16 data\_length, const l\_u8 \*const data)

*This function is used for master node only. It will pack data and send request to slave node with service ID = 0x19: Fault memory read.*

- void [diag\\_fault\\_memory\\_clear](#) (I\_ifc\_handle iii, const I\_u8 NAD, const I\_u8 \*const groupOfDTC)

*This function is used for master node only. It will pack data and send request to slave node with service ID = 0x14: Fault memory clear.*

- void [diag\\_IO\\_control](#) (I\_ifc\_handle iii, const I\_u8 NAD, I\_u16 data\_length, const I\_u8 \*const data)

*This function is used for master node only. It will pack data and send request to slave node with service ID = 0x2F: Input/Output control service.*

- I\_u8 [diag\\_get\\_flag](#) (I\_ifc\_handle iii, I\_u8 flag\_order)

*This function will return flag of diagnostic service, if LIN slave node receive master request of the diagnostic service.*

- void [diag\\_clear\\_flag](#) (I\_ifc\_handle iii, I\_u8 flag\_order)

*This function will clear flag of diagnostic service,.*

## 16.38.2 Function Documentation

### 16.38.2.1 void [diag\\_clear\\_flag](#) ( I\_ifc\_handle iii, I\_u8 flag\_order )

This function will clear flag of diagnostic service,.

#### Parameters

in	iii	LIN interface handle
in	flag_order	Order of service flag

#### Returns

void

Definition at line 1013 of file lin\_diagnostic\_service.c.

### 16.38.2.2 void [diag\\_fault\\_memory\\_clear](#) ( I\_ifc\_handle iii, const I\_u8 NAD, const I\_u8 \*const groupOfDTC )

This function is used for master node only. It will pack data and send request to slave node with service ID = 0x14: Fault memory clear.

#### Parameters

in	iii	LIN interface handle
in	NAD	Node address value of the destination node for the transmission
in	groupOfDTC	contain 3 byte will be transmit follow (byte 0: HighByte, byte 1: Middle Byte, byte 2: Low Byte) to be transmitted

#### Returns

void

Definition at line 759 of file lin\_diagnostic\_service.c.

### 16.38.2.3 void [diag\\_fault\\_memory\\_read](#) ( I\_ifc\_handle iii, const I\_u8 NAD, I\_u16 data\_length, const I\_u8 \*const data )

This function is used for master node only. It will pack data and send request to slave node with service ID = 0x19: Fault memory read.

#### Parameters

in	<i>iii</i>	LIN interface handle
in	<i>NAD</i>	Node address value of the destination node for the transmission
in	<i>data_length</i>	Data length of frame
in	<i>data</i>	Buffer for the data to be transmitted

**Returns**

void

Definition at line 713 of file lin\_diagnostic\_service.c.

**16.38.2.4 I\_u8 diag\_get\_flag ( I\_ifc\_handle *iii*, I\_u8 *flag\_order* )**

This function will return flag of diagnostic service, if LIN slave node receive master request of the diagnostic service.

**Parameters**

in	<i>iii</i>	LIN interface handle
in	<i>flag_order</i>	Order of service flag

**Returns**

1 if LIN Slave node receives master request of the diagnostic service, and the flag has not been cleared by diag\_clear\_flag  
 0 default value  
 0xFF if service is not supported

Definition at line 982 of file lin\_diagnostic\_service.c.

**16.38.2.5 void diag\_IO\_control ( I\_ifc\_handle *iii*, const I\_u8 *NAD*, I\_u16 *data\_length*, const I\_u8 \*const *data* )**

This function is used for master node only. It will pack data and send request to slave node with service ID = 0x2F: Input/Output control service.

**Parameters**

in	<i>iii</i>	LIN interface handle
in	<i>NAD</i>	Node address value of the destination node for the transmission
in	<i>data_length</i>	Data length of frame
in	<i>data</i>	Buffer for the data to be transmitted

**Returns**

void

Definition at line 796 of file lin\_diagnostic\_service.c.

**16.38.2.6 void diag\_read\_data\_by\_identifier ( I\_ifc\_handle *iii*, const I\_u8 *NAD*, const I\_u8 *number\_of\_id*, const I\_u16 \*const *list\_of\_id* )**

This function reads data by identifier, Diagnostic Class II service (0x22).

**Parameters**

in	<i>iii</i>	LIN interface handle
in	<i>NAD</i>	Node address value of the destination node for the transmission

in	<i>number_of_id</i>	number id that send in this request
in	<i>list_of_id</i>	list of id that send in this request

**Returns**

void

This function is for Master node only.

Definition at line 580 of file lin\_diagnostic\_service.c.

**16.38.2.7 void diag\_session\_control ( I\_ifc\_handle *iii*, const I\_u8 *NAD*, const I\_u8 *session\_type* )**

This function is used for master node only. It will pack data and send request to slave node with service ID = 0x10: Session control.

**Parameters**

in	<i>iii</i>	LIN interface handle
in	<i>NAD</i>	Node address value of the destination node for the transmission
in	<i>session_type</i>	is sub function of diagnostic session master request

**Returns**

void

Definition at line 679 of file lin\_diagnostic\_service.c.

**16.38.2.8 void diag\_write\_data\_by\_identifier ( I\_ifc\_handle *iii*, const I\_u8 *NAD*, I\_u16 *data\_length*, const I\_u8 \*const *data* )**

Write Data by Identifier for a specified node - Diagnostic Class II service (0x2E)

**Parameters**

in	<i>iii</i>	Lin interface handle
in	<i>NAD</i>	Node address value of the destination node for the transmission
in	<i>data_length</i>	Data length of frame
in	<i>data</i>	Buffer for the data to be transmitted

**Returns**

void

This function is for Master node only.

Definition at line 629 of file lin\_diagnostic\_service.c.



## 16.39 Driver and cluster management

### 16.39.1 Detailed Description

API perform the initialization of the LIN core.

#### Functions

- `I_bool I_sys_init (void)`

*This function performs the initialization of the LIN core; is the first call a user must use in the LIN core before using any other API functions. The implementation of this function can be replaced by user if needed.*

### 16.39.2 Function Documentation

#### 16.39.2.1 `I_bool I_sys_init ( void )`

This function performs the initialization of the LIN core; is the first call a user must use in the LIN core before using any other API functions. The implementation of this function can be replaced by user if needed.

#### Returns

Operation status = Zero, which is equivalent to 'Initialization was successful'.

Definition at line 57 of file `lin_common_api.c`.

## 16.40 EDMA Driver

### 16.40.1 Detailed Description

This module covers the functionality of the Enhanced Direct Memory Access (eDMA) peripheral driver.

#### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
${S32KSDK_PATH}\platform\drivers\src\edma\edma_driver.c  
${S32KSDK_PATH}\platform\drivers\src\edma\edma_hw_access.c  
${S32KSDK_PATH}\platform\drivers\src\edma\edma_irq.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32KSDK_PATH}\platform\drivers\inc\
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

#### Clock Manager Interrupt Manager (Interrupt)

The eDMA driver implements direct memory access functionality with multiple features: (single block/multi block/loop/scatter-gather transfers); the main usage of this module is to offload the bus read/write accesses from the core to the eDMA engine.

#### Features

- Memory-to-memory, peripheral-to-memory, memory-to-peripheral transfers
- Simple single-block transfers with minimum configuration
- Multi-block transfers with minimum configuration (based on subsequent requests)
- Loop transfers for complex use-cases (e.g. double buffering)
- Scatter/gather
- Dynamic channel allocation

#### Functionality

#### Initialization

In order to use the eDMA driver, the module must be first initialized, using [EDMA\\_DRV\\_Init\(\)](#) function. Once initialized, it cannot be initialized again until it is de-initialized, using [EDMA\\_DRV\\_Deinit\(\)](#). The initialization function does the following operations:

- resets eDMA and DMAMUX modules
- clears the eDMA driver state structure
- sets the arbitration mode and halt settings
- enables error and channel interrupts

Upon module initialization, the application must initialize the channel(s) to be used, using [EDMA\\_DRV\\_ChannelInit\(\)](#) function. This operation means enabling an eDMA channel number (or dynamically allocating one), selecting a source trigger (eDMA request multiplexed via DMAMUX) and setting the channel priority. Additionally, a user callback can be installed for each channel, which will be called when the corresponding interrupt is triggered.

### Transfer Configuration

After initialization, the transfer control descriptor for the selected channel must be configured before use. Depending on the application use-case, one of the three transfer configuration methods should be called.

#### Single-block transfer

For the simplest use-case where a contiguous chunk of data must be transferred, the most suitable function is [EDMA\\_DRV\\_ConfigSingleBlockTransfer\(\)](#). This takes the source/destination addresses as parameters, as well as transfer type/size and data buffer size, and configures the channel TCD to read/write the data in a single request. The looping and scatter/gather features are not used in this scenario. The driver computes the appropriate offsets for source/destination addresses and sets the other TCD fields.

#### Multi-block transfer

This type of transfer can be seen as a sequence of single-block transfers, as described above, which are triggered by subsequent requests. This configuration is suitable for contiguous chunks of data which need to be transferred in multiple steps (e.g. writing one/several bytes from a memory buffer to a peripheral data register each time the module is free - eDMA-based communication). In order to configure this kind of transfer, [EDMA\\_DRV\\_ConfigMultiBlockTransfer\(\)](#) function should be used; aside from the [EDMA\\_DRV\\_ConfigSingleBlockTransfer](#) parameters, this function also takes two additional parameters: the number of transfer loops (expected number of requests to finish the data) and a boolean variable configuring whether requests should be disabled for the current channel upon transfer completion.

#### Loop transfer

The eDMA IP supports complex addressing modes. One of the methods to configure complex transfers in multiple requests is using the minor/major loop support. The [EDMA\\_DRV\\_ConfigLoopTransfer\(\)](#) function sets up the transfer control descriptor for subsequent requests to trigger multiple transfers. The addresses are adjusted after each minor/major loop, according to user setup. This method takes a transfer configuration structure as parameter, with settings for all the fields that control addressing mode (source/destination offsets, minor loop offset, channel linking, minor/major loop count, address last adjustments). It is the responsibility of the application to correctly initialize the configuration structure passed to this function, according to the addressed use-case.

#### Scatter/gather

The eDMA driver also supports scatter/gather feature, which allows various transfer scenarios. When scatter/gather is enabled, a new TCD structure is automatically loaded in the current channel's TCD registers when a transfer is complete, allowing the application to define multiple different subsequent transfers. The [EDMA\\_DRV\\_ConfigScatterGatherTransfer\(\)](#) function sets up a list of TCD structures based on the parameters received and configures the eDMA channel for the first transfer; upon completion, the second TCD from the list will be loaded and the channel will be ready to start the new transfer when a new request is received.

The application must allocate memory for the TCD list passed to this function (with an extra 32-bytes buffer, as the TCD structures need to be 32 bytes aligned); nevertheless, the driver will take care of initializing the array of descriptors, based on the other parameters passed. The function also received two lists of scatter/gather configuration structures (for source and destination, respectively), which define the address, length and type for each transfer. Besides these, the other parameters received are the transfer size, the number of bytes to be transferred on each request and the number of TCD structures to be used. This method will initialize all the descriptors according to user input and link them together; the linkage is done by writing the address of the next descriptor in the appropriate field of each one, similar to a linked-list data structure. The first descriptor is also copied to the TCD registers of the selected channel; if no errors are returned, after calling this function the channel is configured for the transfer defined by the first descriptor.

### Virtual Channel Definition

The virtual channel is used to map multiple hardware channels across multiple eDMA instances. If only one eDMA instance is available, then the virtual channels will map one-on-one with the hardware channels. If more than one eDMA instance is available, then the virtual channels will map continuously and linearly over all of the hardware channels. Example: If the SOC has 4 eDMA modules, each with 32 channels, then the user will be able to address a total of 128 virtual channels, that seamlessly map onto the hardware channels.

### Virtual Channel Control

The eDMA driver provides functions that allow the user to start, stop, allocate and release an eDMA virtual channel. The [EDMA\\_DRV\\_StartChannel\(\)](#) enables the eDMA requests for a virtual channel; this function should be called when the virtual channel is already initialized, as the first request received after the function call will trigger the transfer based on the current values of the virtual channel's TCD registers.

The [EDMA\\_DRV\\_StopChannel\(\)](#) function disables requests for the selected virtual channel; this function should be called whenever the application needs to ignore eDMA requests for a virtual channel. It is automatically called when the virtual channel is released.

The [EDMA\\_DRV\\_SetChannelRequestAndTrigger\(\)](#) function configures the selected virtual channel request and also configures the periodic trigger functionality of the eDMA channel.

Periodic triggering is used by an internal timer to control an eDMA channel.

The [EDMA\\_DRV\\_ReleaseChannel\(\)](#) function frees the hardware and software resources allocated for that virtual channel; it clears the virtual channel state structure, updates the driver state and disables requests for that virtual channel.

### Important Notes

- Before using the eDMA driver the clock for eDMA and DMAMUX modules must be configured
- The driver enables the interrupts for the eDMA module, but any interrupt priority must be done by the application
- When using the modulo feature, application is responsible with ensuring that the source/destination address is properly aligned on a modulo-size boundary.
- The source/destination address must be aligned with transfer size. Ex: With transfer size is 8 bytes, the source/destination address is multiple of 8.
- When using Single-block transfer or Multi-block transfer, NBYTES (Number of bytes to be transferred in each service request of the channel) shall be always configurable on 30 bits instead of 32 bits. This is a limitation only on EDMA Hardware version 2 (S32K1xx platform).
- Limitation of IAR compiler: function alignment is not supported using `ALIGNED()` macro.

### 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 state. Implements : [edma\\_chn\\_state\\_t\\_Class](#). [More...](#)*
- struct [edma\\_channel\\_config\\_t](#)  
*The user configuration structure for the an eDMA driver channel. [More...](#)*
- struct [edma\\_scatter\\_gather\\_list\\_t](#)  
*Data structure for configuring a discrete memory transfer. Implements : [edma\\_scatter\\_gather\\_list\\_t\\_Class](#). [More...](#)*
- struct [edma\\_state\\_t](#)  
*Runtime state structure for the eDMA driver. [More...](#)*
- struct [edma\\_loop\\_transfer\\_config\\_t](#)  
*eDMA loop transfer configuration. [More...](#)*
- struct [edma\\_transfer\\_config\\_t](#)

eDMA transfer size configuration. [More...](#)

- struct [edma\\_software\\_tcd\\_t](#)  
eDMA TCD Implements : [edma\\_software\\_tcd\\_t\\_Class](#) [More...](#)

## Macros

- #define [STCD\\_SIZE](#)(number) (((number) \* 32U) - 1U)  
Macro for the memory size needed for the software TCD.
- #define [STCD\\_ADDR](#)(address) (((uint32\_t)address + 31UL) & ~0x1FUL)
- #define [EDMA\\_ERR\\_LSB\\_MASK](#) 1U  
Macro for accessing the least significant bit of the ERR register.

## Typedefs

- typedef void(\* [edma\\_callback\\_t](#)) (void \*parameter, [edma\\_chn\\_status\\_t](#) status)  
Definition for the eDMA channel callback function.

## Enumerations

- enum [edma\\_channel\\_interrupt\\_t](#) { [EDMA\\_CHN\\_ERR\\_INT](#) = 0U, [EDMA\\_CHN\\_HALF\\_MAJOR\\_LOOP\\_INT](#), [EDMA\\_CHN\\_MAJOR\\_LOOP\\_INT](#) }  
eDMA channel interrupts. Implements : [edma\\_channel\\_interrupt\\_t\\_Class](#)
- enum [edma\\_arbitration\\_algorithm\\_t](#) { [EDMA\\_ARBITRATION\\_FIXED\\_PRIORITY](#) = 0U, [EDMA\\_ARBITRATION\\_ROUND\\_ROBIN](#) }  
eDMA channel arbitration algorithm used for selection among channels. Implements : [edma\\_arbitration\\_algorithm\\_t\\_Class](#)
- enum [edma\\_channel\\_priority\\_t](#) { [EDMA\\_CHN\\_PRIORITY\\_0](#) = 0U, [EDMA\\_CHN\\_PRIORITY\\_1](#) = 1U, [EDMA\\_CHN\\_PRIORITY\\_2](#) = 2U, [EDMA\\_CHN\\_PRIORITY\\_3](#) = 3U, [EDMA\\_CHN\\_PRIORITY\\_4](#) = 4U, [EDMA\\_CHN\\_PRIORITY\\_5](#) = 5U, [EDMA\\_CHN\\_PRIORITY\\_6](#) = 6U, [EDMA\\_CHN\\_PRIORITY\\_7](#) = 7U, [EDMA\\_CHN\\_PRIORITY\\_8](#) = 8U, [EDMA\\_CHN\\_PRIORITY\\_9](#) = 9U, [EDMA\\_CHN\\_PRIORITY\\_10](#) = 10U, [EDMA\\_CHN\\_PRIORITY\\_11](#) = 11U, [EDMA\\_CHN\\_PRIORITY\\_12](#) = 12U, [EDMA\\_CHN\\_PRIORITY\\_13](#) = 13U, [EDMA\\_CHN\\_PRIORITY\\_14](#) = 14U, [EDMA\\_CHN\\_PRIORITY\\_15](#) = 15U, [EDMA\\_CHN\\_DEFAULT\\_PRIORITY](#) = 255U }  
eDMA channel priority setting Implements : [edma\\_channel\\_priority\\_t\\_Class](#)
- enum [edma\\_modulo\\_t](#) { [EDMA\\_MODULO\\_OFF](#) = 0U, [EDMA\\_MODULO\\_2B](#), [EDMA\\_MODULO\\_4B](#), [EDMA\\_MODULO\\_8B](#), [EDMA\\_MODULO\\_16B](#), [EDMA\\_MODULO\\_32B](#), [EDMA\\_MODULO\\_64B](#), [EDMA\\_MODULO\\_128B](#), [EDMA\\_MODULO\\_256B](#), [EDMA\\_MODULO\\_512B](#), [EDMA\\_MODULO\\_1KB](#), [EDMA\\_MODULO\\_2KB](#), [EDMA\\_MODULO\\_4KB](#), [EDMA\\_MODULO\\_8KB](#), [EDMA\\_MODULO\\_16KB](#), [EDMA\\_MODULO\\_32KB](#), [EDMA\\_MODULO\\_64KB](#), [EDMA\\_MODULO\\_128KB](#), [EDMA\\_MODULO\\_256KB](#), [EDMA\\_MODULO\\_512KB](#), [EDMA\\_MODULO\\_1MB](#), [EDMA\\_MODULO\\_2MB](#), [EDMA\\_MODULO\\_4MB](#), [EDMA\\_MODULO\\_8MB](#), [EDMA\\_MODULO\\_16MB](#), [EDMA\\_MODULO\\_32MB](#), [EDMA\\_MODULO\\_64MB](#), [EDMA\\_MODULO\\_128MB](#), [EDMA\\_MODULO\\_256MB](#), [EDMA\\_MODULO\\_512MB](#), [EDMA\\_MODULO\\_1GB](#), [EDMA\\_MODULO\\_2GB](#) }  
eDMA modulo configuration Implements : [edma\\_modulo\\_t\\_Class](#)
- enum [edma\\_transfer\\_size\\_t](#) { [EDMA\\_TRANSFER\\_SIZE\\_1B](#) = 0x0U, [EDMA\\_TRANSFER\\_SIZE\\_2B](#) = 0x1U, [EDMA\\_TRANSFER\\_SIZE\\_4B](#) = 0x2U }  
eDMA transfer configuration Implements : [edma\\_transfer\\_size\\_t\\_Class](#)
- enum [edma\\_chn\\_status\\_t](#) { [EDMA\\_CHN\\_NORMAL](#) = 0U, [EDMA\\_CHN\\_ERROR](#) }  
Channel status for eDMA channel.
- enum [edma\\_transfer\\_type\\_t](#) { [EDMA\\_TRANSFER\\_PERIPH2MEM](#) = 0U, [EDMA\\_TRANSFER\\_MEM2PERIPH](#), [EDMA\\_TRANSFER\\_MEM2MEM](#), [EDMA\\_TRANSFER\\_PERIPH2PERIPH](#) }  
A type for the DMA transfer. Implements : [edma\\_transfer\\_type\\_t\\_Class](#).

### eDMA peripheral driver module level functions

- status\_t [EDMA\\_DRV\\_Init](#) (edma\_state\_t \*edmaState, const edma\_user\_config\_t \*userConfig, edma\_chn\_state\_t \*const chnStateArray[], const edma\_channel\_config\_t \*const chnConfigArray[], uint32\_t chnCount)  
*Initializes the eDMA module.*
- status\_t [EDMA\\_DRV\\_Deinit](#) (void)  
*De-initializes the eDMA module.*

### eDMA peripheral driver channel management functions

- status\_t [EDMA\\_DRV\\_Channellnit](#) (edma\_chn\_state\_t \*edmaChannelState, const edma\_channel\_config\_t \*edmaChannelConfig)  
*Initializes an eDMA channel.*
- status\_t [EDMA\\_DRV\\_ReleaseChannel](#) (uint8\_t virtualChannel)  
*Releases an eDMA channel.*

### eDMA peripheral driver transfer setup functions

- void [EDMA\\_DRV\\_PushConfigToReg](#) (uint8\_t virtualChannel, const edma\_transfer\_config\_t \*tcd)  
*Copies the channel configuration to the TCD registers.*
- void [EDMA\\_DRV\\_PushConfigToSTCD](#) (const edma\_transfer\_config\_t \*config, edma\_software\_tcd\_t \*stcd)  
*Copies the channel configuration to the software TCD structure.*
- status\_t [EDMA\\_DRV\\_ConfigSingleBlockTransfer](#) (uint8\_t virtualChannel, edma\_transfer\_type\_t type, uint32\_t srcAddr, uint32\_t destAddr, edma\_transfer\_size\_t transferSize, uint32\_t dataBufferSize)  
*Configures a simple single block data transfer with DMA.*
- status\_t [EDMA\\_DRV\\_ConfigMultiBlockTransfer](#) (uint8\_t virtualChannel, edma\_transfer\_type\_t type, uint32\_t srcAddr, uint32\_t destAddr, edma\_transfer\_size\_t transferSize, uint32\_t blockSize, uint32\_t blockCount, bool disableReqOnCompletion)  
*Configures a multiple block data transfer with DMA.*
- status\_t [EDMA\\_DRV\\_ConfigLoopTransfer](#) (uint8\_t virtualChannel, const edma\_transfer\_config\_t \*transferConfig)  
*Configures the DMA transfer in loop mode.*
- status\_t [EDMA\\_DRV\\_ConfigScatterGatherTransfer](#) (uint8\_t virtualChannel, edma\_software\_tcd\_t \*stcd, edma\_transfer\_size\_t transferSize, uint32\_t bytesOnEachRequest, const edma\_scatter\_gather\_list\_t \*srcList, const edma\_scatter\_gather\_list\_t \*destList, uint8\_t tcdCount)  
*Configures the DMA transfer in a scatter-gather mode.*
- void [EDMA\\_DRV\\_CancelTransfer](#) (bool error)  
*Cancel the running transfer.*

### eDMA Peripheral driver channel operation functions

- status\_t [EDMA\\_DRV\\_StartChannel](#) (uint8\_t virtualChannel)  
*Starts an eDMA channel.*
- status\_t [EDMA\\_DRV\\_StopChannel](#) (uint8\_t virtualChannel)  
*Stops the eDMA channel.*
- status\_t [EDMA\\_DRV\\_SetChannelRequestAndTrigger](#) (uint8\_t virtualChannel, uint8\_t request, bool enableTrigger)  
*Configures the DMA request for the eDMA channel.*
- void [EDMA\\_DRV\\_ClearTCD](#) (uint8\_t virtualChannel)  
*Clears all registers to 0 for the channel's TCD.*
- void [EDMA\\_DRV\\_SetSrcAddr](#) (uint8\_t virtualChannel, uint32\_t address)  
*Configures the source address for the eDMA channel.*

- void [EDMA\\_DRV\\_SetSrcOffset](#) (uint8\_t virtualChannel, int16\_t offset)  
*Configures the source address signed offset for the eDMA channel.*
- void [EDMA\\_DRV\\_SetSrcReadChunkSize](#) (uint8\_t virtualChannel, [edma\\_transfer\\_size\\_t](#) size)  
*Configures the source data chunk size (transferred in a read sequence).*
- void [EDMA\\_DRV\\_SetSrcLastAddrAdjustment](#) (uint8\_t virtualChannel, int32\_t adjust)  
*Configures the source address last adjustment.*
- void [EDMA\\_DRV\\_SetDestAddr](#) (uint8\_t virtualChannel, uint32\_t address)  
*Configures the destination address for the eDMA channel.*
- void [EDMA\\_DRV\\_SetDestOffset](#) (uint8\_t virtualChannel, int16\_t offset)  
*Configures the destination address signed offset for the eDMA channel.*
- void [EDMA\\_DRV\\_SetDestWriteChunkSize](#) (uint8\_t virtualChannel, [edma\\_transfer\\_size\\_t](#) size)  
*Configures the destination data chunk size (transferred in a write sequence).*
- void [EDMA\\_DRV\\_SetDestLastAddrAdjustment](#) (uint8\_t virtualChannel, int32\_t adjust)  
*Configures the destination address last adjustment.*
- void [EDMA\\_DRV\\_SetMinorLoopBlockSize](#) (uint8\_t virtualChannel, uint32\_t nbytes)  
*Configures the number of bytes to be transferred in each service request of the channel.*
- void [EDMA\\_DRV\\_SetMajorLoopIterationCount](#) (uint8\_t virtualChannel, uint32\_t majorLoopCount)  
*Configures the number of major loop iterations.*
- uint32\_t [EDMA\\_DRV\\_GetRemainingMajorIterationsCount](#) (uint8\_t virtualChannel)  
*Returns the remaining major loop iteration count.*
- void [EDMA\\_DRV\\_SetScatterGatherLink](#) (uint8\_t virtualChannel, uint32\_t nextTCDAddr)  
*Configures the memory address of the next TCD, in scatter/gather mode.*
- void [EDMA\\_DRV\\_DisableRequestsOnTransferComplete](#) (uint8\_t virtualChannel, bool disable)  
*Disables/Enables the DMA request after the major loop completes for the TCD.*
- void [EDMA\\_DRV\\_ConfigureInterrupt](#) (uint8\_t virtualChannel, [edma\\_channel\\_interrupt\\_t](#) intSrc, bool enable)  
*Disables/Enables the channel interrupt requests.*
- void [EDMA\\_DRV\\_TriggerSwRequest](#) (uint8\_t virtualChannel)  
*Triggers a sw request for the current channel.*

#### eDMA Peripheral callback and interrupt functions

- status\_t [EDMA\\_DRV\\_InstallCallback](#) (uint8\_t virtualChannel, [edma\\_callback\\_t](#) callback, void \*parameter)  
*Registers the callback function and the parameter for eDMA channel.*

#### eDMA Peripheral driver miscellaneous functions

- [edma\\_chn\\_status\\_t](#) [EDMA\\_DRV\\_GetChannelStatus](#) (uint8\_t virtualChannel)  
*Gets the eDMA channel status.*

### 16.40.2 Data Structure Documentation

#### 16.40.2.1 struct edma\_user\_config\_t

The user configuration structure for the eDMA driver.

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. Implements : [edma\\_user\\_config\\_t\\_Class](#)

Definition at line 232 of file [edma\\_driver.h](#).

#### Data Fields

- [edma\\_arbitration\\_algorithm\\_t](#) chnArbitration
- bool haltOnError

## Field Documentation

### 16.40.2.1.1 `edma_arbitration_algorithm_t` `chnArbitration`

eDMA channel arbitration.

Definition at line 233 of file `edma_driver.h`.

### 16.40.2.1.2 `bool` `haltOnError`

Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.

Definition at line 241 of file `edma_driver.h`.

### 16.40.2.2 `struct edma_chn_state_t`

Data structure for the eDMA channel state. Implements : `edma_chn_state_t_Class`.

Definition at line 268 of file `edma_driver.h`.

## Data Fields

- `uint8_t` `virtChn`
- `edma_callback_t` `callback`
- `void *` `parameter`
- `volatile edma_chn_status_t` `status`

## Field Documentation

### 16.40.2.2.1 `edma_callback_t` `callback`

Callback function pointer for the eDMA channel. It will be called at the eDMA channel complete and eDMA channel error.

Definition at line 270 of file `edma_driver.h`.

### 16.40.2.2.2 `void*` `parameter`

Parameter for the callback function pointer.

Definition at line 273 of file `edma_driver.h`.

### 16.40.2.2.3 `volatile edma_chn_status_t` `status`

eDMA channel status.

Definition at line 274 of file `edma_driver.h`.

### 16.40.2.2.4 `uint8_t` `virtChn`

Virtual channel number.

Definition at line 269 of file `edma_driver.h`.

### 16.40.2.3 `struct edma_channel_config_t`

The user configuration structure for the an eDMA driver channel.

Use an instance of this structure with the `EDMA_DRV_ChannelInit()` function. This allows the user to configure settings of the EDMA channel with a single function call. Implements : `edma_channel_config_t_Class`

Definition at line 284 of file `edma_driver.h`.

## Data Fields

- `edma_channel_priority_t` `channelPriority`



- uint8\_t [virtChnConfig](#)
- [edma\\_callback\\_t](#) callback
- void \* [callbackParam](#)
- bool [enableTrigger](#)

#### Field Documentation

##### 16.40.2.3.1 [edma\\_callback\\_t](#) callback

Callback that will be registered for this channel

Definition at line 297 of file `edma_driver.h`.

##### 16.40.2.3.2 void\* [callbackParam](#)

Parameter passed to the channel callback

Definition at line 298 of file `edma_driver.h`.

##### 16.40.2.3.3 [edma\\_channel\\_priority\\_t](#) channelPriority

eDMA channel priority - only used when channel arbitration mode is 'Fixed priority'.

Definition at line 291 of file `edma_driver.h`.

##### 16.40.2.3.4 bool [enableTrigger](#)

Enables the periodic trigger capability for the DMA channel.

Definition at line 299 of file `edma_driver.h`.

##### 16.40.2.3.5 uint8\_t [virtChnConfig](#)

eDMA virtual channel number

Definition at line 293 of file `edma_driver.h`.

##### 16.40.2.4 struct [edma\\_scatter\\_gather\\_list\\_t](#)

Data structure for configuring a discrete memory transfer. Implements : `edma_scatter_gather_list_t_Class`.

Definition at line 315 of file `edma_driver.h`.

#### Data Fields

- uint32\_t [address](#)
- uint32\_t [length](#)
- [edma\\_transfer\\_type\\_t](#) type

#### Field Documentation

##### 16.40.2.4.1 uint32\_t [address](#)

Address of buffer.

Definition at line 316 of file `edma_driver.h`.

##### 16.40.2.4.2 uint32\_t [length](#)

Length of buffer.

Definition at line 317 of file `edma_driver.h`.

##### 16.40.2.4.3 [edma\\_transfer\\_type\\_t](#) type

Type of the DMA transfer

Definition at line 318 of file `edma_driver.h`.

#### 16.40.2.5 `struct edma_state_t`

Runtime state structure for the eDMA driver.

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. Implements : `edma_state_t_Class`

Definition at line 330 of file `edma_driver.h`.

#### Data Fields

- `edma_chn_state_t` \*volatile `virtChnState` [(uint32\_t) FEATURE\_DMA\_VIRTUAL\_CHANNELS]

#### Field Documentation

##### 16.40.2.5.1 `edma_chn_state_t`\* volatile `virtChnState`[(uint32\_t) FEATURE\_DMA\_VIRTUAL\_CHANNELS]

Pointer array storing channel state.

Definition at line 331 of file `edma_driver.h`.

#### 16.40.2.6 `struct edma_loop_transfer_config_t`

eDMA loop transfer configuration.

This structure configures the basic minor/major loop attributes. Implements : `edma_loop_transfer_config_t_Class`

Definition at line 340 of file `edma_driver.h`.

#### Data Fields

- uint32\_t `majorLoopIterationCount`
- bool `srcOffsetEnable`
- bool `dstOffsetEnable`
- int32\_t `minorLoopOffset`
- bool `minorLoopChnLinkEnable`
- uint8\_t `minorLoopChnLinkNumber`
- bool `majorLoopChnLinkEnable`
- uint8\_t `majorLoopChnLinkNumber`

#### Field Documentation

##### 16.40.2.6.1 `bool dstOffsetEnable`

Selects whether the minor loop offset is applied to the destination address upon minor loop completion.

Definition at line 344 of file `edma_driver.h`.

##### 16.40.2.6.2 `bool majorLoopChnLinkEnable`

Enables channel-to-channel linking on major loop complete.

Definition at line 351 of file `edma_driver.h`.

##### 16.40.2.6.3 `uint8_t majorLoopChnLinkNumber`

The number of the next channel to be started by DMA engine when major loop completes.

Definition at line 352 of file `edma_driver.h`.

**16.40.2.6.4 uint32\_t majorLoopIterationCount**

Number of major loop iterations.

Definition at line 341 of file edma\_driver.h.

**16.40.2.6.5 bool minorLoopChnLinkEnable**

Enables channel-to-channel linking on minor loop complete.

Definition at line 348 of file edma\_driver.h.

**16.40.2.6.6 uint8\_t minorLoopChnLinkNumber**

The number of the next channel to be started by DMA engine when minor loop completes.

Definition at line 349 of file edma\_driver.h.

**16.40.2.6.7 int32\_t minorLoopOffset**

Sign-extended offset applied to the source or destination address to form the next-state value after the minor loop completes.

Definition at line 346 of file edma\_driver.h.

**16.40.2.6.8 bool srcOffsetEnable**

Selects whether the minor loop offset is applied to the source address upon minor loop completion.

Definition at line 342 of file edma\_driver.h.

**16.40.2.7 struct edma\_transfer\_config\_t**

eDMA transfer size configuration.

This structure configures the basic source/destination transfer attribute. Implements : edma\_transfer\_config\_t ↔ Class

Definition at line 362 of file edma\_driver.h.

**Data Fields**

- uint32\_t srcAddr
- uint32\_t destAddr
- edma\_transfer\_size\_t srcTransferSize
- edma\_transfer\_size\_t destTransferSize
- int16\_t srcOffset
- int16\_t destOffset
- int32\_t srcLastAddrAdjust
- int32\_t destLastAddrAdjust
- edma\_modulo\_t srcModulo
- edma\_modulo\_t destModulo
- uint32\_t minorByteTransferCount
- bool scatterGatherEnable
- uint32\_t scatterGatherNextDescAddr
- bool interruptEnable
- edma\_loop\_transfer\_config\_t \* loopTransferConfig

**Field Documentation****16.40.2.7.1 uint32\_t destAddr**

Memory address pointing to the destination data.

Definition at line 364 of file edma\_driver.h.

**16.40.2.7.2 int32\_t destLastAddrAdjust**

Last destination address adjustment. Note here it is only valid when scatter/gather feature is not enabled.

Definition at line 374 of file edma\_driver.h.

**16.40.2.7.3 edma\_modulo\_t destModulo**

Destination address modulo.

Definition at line 377 of file edma\_driver.h.

**16.40.2.7.4 int16\_t destOffset**

Sign-extended offset applied to the current destination address to form the next-state value as each source read/write is completed.

Definition at line 370 of file edma\_driver.h.

**16.40.2.7.5 edma\_transfer\_size\_t destTransferSize**

Destination data transfer size.

Definition at line 366 of file edma\_driver.h.

**16.40.2.7.6 bool interruptEnable**

Enable the interrupt request when the major loop count completes

Definition at line 385 of file edma\_driver.h.

**16.40.2.7.7 edma\_loop\_transfer\_config\_t\* loopTransferConfig**

Pointer to loop transfer configuration structure (defines minor/major loop attributes) Note: this field is only used when minor loop mapping is enabled from DMA configuration.

Definition at line 387 of file edma\_driver.h.

**16.40.2.7.8 uint32\_t minorByteTransferCount**

Number of bytes to be transferred in each service request of the channel.

Definition at line 378 of file edma\_driver.h.

**16.40.2.7.9 bool scatterGatherEnable**

Enable scatter gather feature.

Definition at line 380 of file edma\_driver.h.

**16.40.2.7.10 uint32\_t scatterGatherNextDescAddr**

The address of the next descriptor to be used, when scatter/gather feature is enabled. Note: this value is not used when scatter/gather feature is disabled.

Definition at line 381 of file edma\_driver.h.

**16.40.2.7.11 uint32\_t srcAddr**

Memory address pointing to the source data.

Definition at line 363 of file edma\_driver.h.

**16.40.2.7.12 int32\_t srcLastAddrAdjust**

Last source address adjustment.

Definition at line 373 of file edma\_driver.h.

#### 16.40.2.7.13 edma\_modulo\_t srcModulo

Source address modulo.

Definition at line 376 of file edma\_driver.h.

#### 16.40.2.7.14 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.

Definition at line 367 of file edma\_driver.h.

#### 16.40.2.7.15 edma\_transfer\_size\_t srcTransferSize

Source data transfer size.

Definition at line 365 of file edma\_driver.h.

#### 16.40.2.8 struct edma\_software\_tcd\_t

eDMA TCD Implements : edma\_software\_tcd\_t\_Class

Definition at line 397 of file edma\_driver.h.

##### Data Fields

- uint32\_t [SADDR](#)
- int16\_t [SOFF](#)
- uint16\_t [ATTR](#)
- uint32\_t [NBYTES](#)
- int32\_t [SLAST](#)
- uint32\_t [DADDR](#)
- int16\_t [DOFF](#)
- uint16\_t [CITER](#)
- int32\_t [DLAST\\_SGA](#)
- uint16\_t [CSR](#)
- uint16\_t [BITER](#)

##### Field Documentation

#### 16.40.2.8.1 uint16\_t ATTR

Definition at line 400 of file edma\_driver.h.

#### 16.40.2.8.2 uint16\_t BITER

Definition at line 408 of file edma\_driver.h.

#### 16.40.2.8.3 uint16\_t CITER

Definition at line 405 of file edma\_driver.h.

#### 16.40.2.8.4 uint16\_t CSR

Definition at line 407 of file edma\_driver.h.

#### 16.40.2.8.5 uint32\_t DADDR

Definition at line 403 of file edma\_driver.h.

**16.40.2.8.6 int32\_t DLAST\_SGA**

Definition at line 406 of file edma\_driver.h.

**16.40.2.8.7 int16\_t DOFF**

Definition at line 404 of file edma\_driver.h.

**16.40.2.8.8 uint32\_t NBYTES**

Definition at line 401 of file edma\_driver.h.

**16.40.2.8.9 uint32\_t SADDR**

Definition at line 398 of file edma\_driver.h.

**16.40.2.8.10 int32\_t SLAST**

Definition at line 402 of file edma\_driver.h.

**16.40.2.8.11 int16\_t SOFF**

Definition at line 399 of file edma\_driver.h.

**16.40.3 Macro Definition Documentation****16.40.3.1 #define EDMA\_ERR\_LSB\_MASK 1U**

Macro for accessing the least significant bit of the ERR register.

The erroneous channels are retrieved from ERR register by subsequently right shifting all the ERR bits + "AND"-ing the result with this mask.

Definition at line 65 of file edma\_driver.h.

**16.40.3.2 #define STCD\_ADDR( address ) (((uint32\_t)address + 31UL) & ~0x1FUL)**

Definition at line 57 of file edma\_driver.h.

**16.40.3.3 #define STCD\_SIZE( number ) (((number) \* 32U) - 1U)**

Macro for the memory size needed for the software TCD.

Software TCD is aligned to 32 bytes. We don't need a software TCD structure for the first descriptor, since the configuration is pushed directly to registers. To make sure the software TCD can meet the eDMA module requirement regarding alignment, allocate memory for the remaining descriptors with extra 31 bytes.

Definition at line 56 of file edma\_driver.h.

**16.40.4 Typedef Documentation****16.40.4.1 typedef void(\* edma\_callback\_t)(void \*parameter, edma\_chn\_status\_t status)**

Definition for the eDMA channel callback function.

Prototype for the callback function registered in the eDMA driver. Implements : edma\_callback\_t\_Class

Definition at line 263 of file edma\_driver.h.

**16.40.5 Enumeration Type Documentation**

16.40.5.1 enum `edma_arbitration_algorithm_t`

eDMA channel arbitration algorithm used for selection among channels. Implements : `edma_arbitration_algorithm_t_Class`

Enumerator

**`EDMA_ARBITRATION_FIXED_PRIORITY`** Fixed Priority

**`EDMA_ARBITRATION_ROUND_ROBIN`** Round-Robin arbitration

Definition at line 79 of file `edma_driver.h`.

16.40.5.2 enum `edma_channel_interrupt_t`

eDMA channel interrupts. Implements : `edma_channel_interrupt_t_Class`

Enumerator

**`EDMA_CHN_ERR_INT`** Error interrupt

**`EDMA_CHN_HALF_MAJOR_LOOP_INT`** Half major loop interrupt.

**`EDMA_CHN_MAJOR_LOOP_INT`** Complete major loop interrupt.

Definition at line 70 of file `edma_driver.h`.

16.40.5.3 enum `edma_channel_priority_t`

eDMA channel priority setting Implements : `edma_channel_priority_t_Class`

Enumerator

**`EDMA_CHN_PRIORITY_0`**

**`EDMA_CHN_PRIORITY_1`**

**`EDMA_CHN_PRIORITY_2`**

**`EDMA_CHN_PRIORITY_3`**

**`EDMA_CHN_PRIORITY_4`**

**`EDMA_CHN_PRIORITY_5`**

**`EDMA_CHN_PRIORITY_6`**

**`EDMA_CHN_PRIORITY_7`**

**`EDMA_CHN_PRIORITY_8`**

**`EDMA_CHN_PRIORITY_9`**

**`EDMA_CHN_PRIORITY_10`**

**`EDMA_CHN_PRIORITY_11`**

**`EDMA_CHN_PRIORITY_12`**

**`EDMA_CHN_PRIORITY_13`**

**`EDMA_CHN_PRIORITY_14`**

**`EDMA_CHN_PRIORITY_15`**

**`EDMA_CHN_DEFAULT_PRIORITY`**

Definition at line 87 of file `edma_driver.h`.

#### 16.40.5.4 enum edma\_chn\_status\_t

Channel status for eDMA channel.

A structure describing the eDMA channel status. The user can get the status by callback parameter or by calling EDMA\_DRV\_getStatus() function. Implements : edma\_chn\_status\_t\_Class

Enumerator

**EDMA\_CHN\_NORMAL** eDMA channel normal state.

**EDMA\_CHN\_ERROR** An error occurred in the eDMA channel.

Definition at line 252 of file edma\_driver.h.

#### 16.40.5.5 enum edma\_modulo\_t

eDMA modulo configuration Implements : edma\_modulo\_t\_Class

Enumerator

**EDMA\_MODULO\_OFF**

**EDMA\_MODULO\_2B**

**EDMA\_MODULO\_4B**

**EDMA\_MODULO\_8B**

**EDMA\_MODULO\_16B**

**EDMA\_MODULO\_32B**

**EDMA\_MODULO\_64B**

**EDMA\_MODULO\_128B**

**EDMA\_MODULO\_256B**

**EDMA\_MODULO\_512B**

**EDMA\_MODULO\_1KB**

**EDMA\_MODULO\_2KB**

**EDMA\_MODULO\_4KB**

**EDMA\_MODULO\_8KB**

**EDMA\_MODULO\_16KB**

**EDMA\_MODULO\_32KB**

**EDMA\_MODULO\_64KB**

**EDMA\_MODULO\_128KB**

**EDMA\_MODULO\_256KB**

**EDMA\_MODULO\_512KB**

**EDMA\_MODULO\_1MB**

**EDMA\_MODULO\_2MB**

**EDMA\_MODULO\_4MB**

**EDMA\_MODULO\_8MB**

**EDMA\_MODULO\_16MB**

**EDMA\_MODULO\_32MB**

**EDMA\_MODULO\_64MB**

**EDMA\_MODULO\_128MB**

**EDMA\_MODULO\_256MB**

**EDMA\_MODULO\_512MB**

**EDMA\_MODULO\_1GB**

**EDMA\_MODULO\_2GB**

Definition at line 159 of file edma\_driver.h.



16.40.5.6 enum `edma_transfer_size_t`

eDMA transfer configuration Implements : `edma_transfer_size_t_Class`

Enumerator

**`EDMA_TRANSFER_SIZE_1B`**

**`EDMA_TRANSFER_SIZE_2B`**

**`EDMA_TRANSFER_SIZE_4B`**

Definition at line 197 of file `edma_driver.h`.

16.40.5.7 enum `edma_transfer_type_t`

A type for the DMA transfer. Implements : `edma_transfer_type_t_Class`.

Enumerator

**`EDMA_TRANSFER_PERIPH2MEM`** Transfer from peripheral to memory

**`EDMA_TRANSFER_MEM2PERIPH`** Transfer from memory to peripheral

**`EDMA_TRANSFER_MEM2MEM`** Transfer from memory to memory

**`EDMA_TRANSFER_PERIPH2PERIPH`** Transfer from peripheral to peripheral

Definition at line 305 of file `edma_driver.h`.

## 16.40.6 Function Documentation

16.40.6.1 void `EDMA_DRV_CancelTransfer ( bool error )`

Cancel the running transfer.

This function cancels the current transfer, optionally signalling an error.

Parameters

<i>bool</i>	error If true, an error will be logged for the current transfer.
-------------	--

Definition at line 1487 of file `edma_driver.c`.

16.40.6.2 status\_t `EDMA_DRV_Channellnit ( edma_chn_state_t * edmaChannelState, const edma_channel_config_t * edmaChannelConfig )`

Initializes an eDMA channel.

This function initializes the run-time state structure for a eDMA channel, based on user configuration. It will request the channel, set up the channel priority and install the callback.

Parameters

<i>edmaChannelState</i>	Pointer to the eDMA channel 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 channel status. The memory must be kept valid before calling the <code>EDMA_DRV_ReleaseChannel</code> .
-------------------------	---

<i>edmaChannel</i> ↔ <i>Config</i>	User configuration structure for eDMA channel. The user populates the members of this structure and passes the pointer of this structure into the function.
---------------------------------------	---

**Returns**

STATUS\_ERROR or STATUS\_SUCCESS.

Definition at line 287 of file edma\_driver.c.

#### 16.40.6.3 void EDMA\_DRV\_ClearTCD ( uint8\_t *virtualChannel* )

Clears all registers to 0 for the channel's TCD.

**Parameters**

<i>virtualChannel</i>	eDMA virtual channel number.
-----------------------	------------------------------

Definition at line 1019 of file edma\_driver.c.

#### 16.40.6.4 status\_t EDMA\_DRV\_ConfigLoopTransfer ( uint8\_t *virtualChannel*, const edma\_transfer\_config\_t \* *transferConfig* )

Configures the DMA transfer in loop mode.

This function configures the DMA transfer in a loop chain. The user passes a block of memory into this function that configures the loop transfer properties (minor/major loop count, address offsets, channel linking). The DMA driver copies the configuration to TCD registers, only when the loop properties are set up correctly and minor loop mapping is enabled for the eDMA module.

**Parameters**

<i>virtualChannel</i>	eDMA virtual channel number.
<i>transferConfig</i>	Pointer to the transfer configuration structure; this structure defines fields for setting up the basic transfer and also a pointer to a memory structure that defines the loop chain properties (minor/major).

**Returns**

STATUS\_ERROR or STATUS\_SUCCESS

Definition at line 689 of file edma\_driver.c.

#### 16.40.6.5 status\_t EDMA\_DRV\_ConfigMultiBlockTransfer ( uint8\_t *virtualChannel*, edma\_transfer\_type\_t *type*, uint32\_t *srcAddr*, uint32\_t *destAddr*, edma\_transfer\_size\_t *transferSize*, uint32\_t *blockSize*, uint32\_t *blockCount*, bool *disableReqOnCompletion* )

Configures a multiple block data transfer with DMA.

This function configures the descriptor for a multi-block transfer. The function considers contiguous memory blocks, thus it configures the TCD source/destination offset fields to cover the data buffer without gaps, according to "transferSize" parameter (the offset is equal to the number of bytes transferred in a source read/destination write). The buffer is divided in multiple block, each block being transferred upon a single DMA request.

NOTE: For transfers to/from peripherals, make sure the transfer size is equal to the data buffer size of the peripheral used, otherwise only truncated chunks of data may be transferred (e.g. for a communication IP with an 8-bit data register the transfer size should be 1B, whereas for a 32-bit data register, the transfer size should be 4B). The rationale of this constraint is that, on the peripheral side, the address offset is set to zero, allowing to read/write data from/to the peripheral in a single source read/destination write operation.

## Parameters

<i>virtualChannel</i>	eDMA virtual channel number.
<i>type</i>	Transfer type (M->M, P->M, M->P, P->P).
<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>transferSize</i>	The number of bytes to be transferred on every DMA write/read. Source/Dest share the same write/read size.
<i>blockSize</i>	The total number of bytes inside a block.
<i>blockCount</i>	The total number of data blocks (one block is transferred upon a DMA request).
<i>disableReqOnCompletion</i>	This parameter specifies whether the DMA channel should be disabled when the transfer is complete (further requests will remain untreated).

## Returns

STATUS\_ERROR or STATUS\_SUCCESS

Definition at line 629 of file edma\_driver.c.

**16.40.6.6** `status_t EDMA_DRV_ConfigScatterGatherTransfer ( uint8_t virtualChannel, edma_software_tcd_t * stcd, edma_transfer_size_t transferSize, uint32_t bytesOnEachRequest, const edma_scatter_gather_list_t * srcList, const edma_scatter_gather_list_t * destList, uint8_t tcdCount )`

Configures the DMA transfer in a scatter-gather mode.

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>virtualChannel</i>	eDMA virtual channel number.
<i>stcd</i>	Array of empty software TCD structures. The user must prepare this memory block. We don't need a software TCD structure for the first descriptor, since the configuration is pushed directly to registers. The "stcd" buffer must align with 32 bytes; if not, an error occurs in the eDMA driver. Thus, the required memory size for "stcd" is equal to <code>tcdCount * size_of(edma_software_tcd_t) - 1</code> ; the driver will take care of the memory alignment if the provided memory buffer is big enough. For proper allocation of the "stcd" buffer it is recommended to use <code>STCD_SIZE</code> macro.
<i>transferSize</i>	The number of bytes to be transferred on every DMA write/read.
<i>bytesOnEachRequest</i>	Bytes to be transferred in each DMA request.
<i>srcList</i>	Data structure storing the address, length and type of transfer (M->M, M->P, P->M, P->P) for the bytes 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, length and type of transfer (M->M, M->P, P->M, P->P) for the bytes to be transferred for destination memory blocks. 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>tcdCount</i>	The number of TCD memory blocks contained in the scatter gather list.
-----------------	---

**Returns**

STATUS\_ERROR or STATUS\_SUCCESS

Definition at line 759 of file edma\_driver.c.

16.40.6.7 **status\_t** EDMA\_DRV\_ConfigSingleBlockTransfer ( **uint8\_t** *virtualChannel*, **edma\_transfer\_type\_t** *type*, **uint32\_t** *srcAddr*, **uint32\_t** *destAddr*, **edma\_transfer\_size\_t** *transferSize*, **uint32\_t** *dataBufferSize* )

Configures a simple single block data transfer with DMA.

This function configures the descriptor for a single block transfer. The function considers contiguous memory blocks, thus it configures the TCD source/destination offset fields to cover the data buffer without gaps, according to "transferSize" parameter (the offset is equal to the number of bytes transferred in a source read/destination write).

NOTE: For memory-to-peripheral or peripheral-to-memory transfers, make sure the transfer size is equal to the data buffer size of the peripheral used, otherwise only truncated chunks of data may be transferred (e.g. for a communication IP with an 8-bit data register the transfer size should be 1B, whereas for a 32-bit data register, the transfer size should be 4B). The rationale of this constraint is that, on the peripheral side, the address offset is set to zero, allowing to read/write data from/to the peripheral in a single source read/destination write operation.

**Parameters**

<i>virtualChannel</i>	eDMA virtual channel number.
<i>type</i>	Transfer type (M->M, P->M, M->P, P->P).
<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>transferSize</i>	The number of bytes to be transferred on every DMA write/read. Source/Dest share the same write/read size.
<i>dataBufferSize</i>	The total number of bytes to be transferred.

**Returns**

STATUS\_ERROR or STATUS\_SUCCESS

Definition at line 509 of file edma\_driver.c.

16.40.6.8 **void** EDMA\_DRV\_ConfigureInterrupt ( **uint8\_t** *virtualChannel*, **edma\_channel\_interrupt\_t** *intSrc*, **bool** *enable* )

Disables/Enables the channel interrupt requests.

This function enables/disables error, half major loop and complete major loop interrupts for the current channel.

**Parameters**

<i>virtualChannel</i>	eDMA virtual channel number.
<i>interrupt</i>	Interrupt event (error/half major loop/complete major loop).
<i>enable</i>	Enable (true)/Disable (false) interrupts for the current channel.

Definition at line 1439 of file edma\_driver.c.

16.40.6.9 **status\_t** EDMA\_DRV\_Deinit ( **void** )

De-initializes the eDMA module.

This function resets the eDMA module to reset state and disables the interrupt to the core.

**Returns**

STATUS\_ERROR or STATUS\_SUCCESS.

Definition at line 239 of file edma\_driver.c.

16.40.6.10 void EDMA\_DRV\_DisableRequestsOnTransferComplete ( uint8\_t *virtualChannel*, bool *disable* )

Disables/Enables the DMA request after the major loop completes for the TCD.

If disabled, the eDMA hardware automatically clears the corresponding DMA request when the current major iteration count reaches zero.

#### Parameters

<i>virtualChannel</i>	eDMA virtual channel number.
<i>disable</i>	Disable (true)/Enable (false) DMA request after TCD complete.

Definition at line 1409 of file edma\_driver.c.

16.40.6.11 edma\_chn\_status\_t EDMA\_DRV\_GetChannelStatus ( uint8\_t *virtualChannel* )

Gets the eDMA channel status.

#### Parameters

<i>virtualChannel</i>	eDMA virtual channel number.
-----------------------	------------------------------

#### Returns

Channel status.

Definition at line 1711 of file edma\_driver.c.

16.40.6.12 uint32\_t EDMA\_DRV\_GetRemainingMajorIterationsCount ( uint8\_t *virtualChannel* )

Returns the remaining major loop iteration count.

Gets the number minor loops yet to be triggered (major loop iterations).

#### Parameters

<i>virtualChannel</i>	eDMA virtual channel number.
-----------------------	------------------------------

#### Returns

number of major loop iterations yet to be triggered

Definition at line 1348 of file edma\_driver.c.

16.40.6.13 status\_t EDMA\_DRV\_Init ( edma\_state\_t \* *edmaState*, const edma\_user\_config\_t \* *userConfig*, edma\_chn\_state\_t \*const *chnStateArray*[], const edma\_channel\_config\_t \*const *chnConfigArray*[], uint32\_t *chnCount* )

Initializes the eDMA module.

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 resets the eDMA modules, initializes the module to user-defined settings and default settings.

#### 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 EDMA_DRV_DeInit.
------------------	--

<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.
<i>chnStateArray</i>	Array of pointers to run-time state structures for eDMA channels; will populate the state structures inside the eDMA driver state structure.
<i>chnConfigArray</i>	Array of pointers to channel initialization structures.
<i>chnCount</i>	The number of eDMA channels to be initialized.

**Returns**

STATUS\_ERROR or STATUS\_SUCCESS.

Definition at line 123 of file edma\_driver.c.

**16.40.6.14** `status_t EDMA_DRV_InstallCallback ( uint8_t virtualChannel, edma_callback_t callback, void * parameter )`

Registers the callback function and the parameter for eDMA channel.

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>virtualChannel</i>	eDMA virtual channel number.
<i>callback</i>	The pointer to the callback function.
<i>parameter</i>	The pointer to the callback function's parameter.

**Returns**

STATUS\_ERROR or STATUS\_SUCCESS.

Definition at line 365 of file edma\_driver.c.

**16.40.6.15** `void EDMA_DRV_PushConfigToReg ( uint8_t virtualChannel, const edma_transfer_config_t * tcd )`

Copies the channel configuration to the TCD registers.

**Parameters**

<i>virtualChannel</i>	eDMA virtual channel number.
<i>tcd</i>	Pointer to the channel configuration structure.

Definition at line 1589 of file edma\_driver.c.

**16.40.6.16** `void EDMA_DRV_PushConfigToSTCD ( const edma_transfer_config_t * config, edma_software_tcd_t * stcd )`

Copies the channel configuration to the software TCD structure.

This function copies the properties from the channel configuration to the software TCD structure; the address of the software TCD can be used to enable scatter/gather operation (pointer to the next TCD).

**Parameters**

<i>config</i>	Pointer to the channel configuration structure.
---------------	---

<i>stcd</i>	Pointer to the software TCD structure.
-------------	--

Definition at line 1545 of file edma\_driver.c.

**16.40.6.17** `status_t EDMA_DRV_ReleaseChannel ( uint8_t virtualChannel )`

Releases an eDMA channel.

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>virtualChannel</i>	eDMA virtual channel number.
-----------------------	------------------------------

#### Returns

STATUS\_ERROR or STATUS\_SUCCESS.

Definition at line 391 of file edma\_driver.c.

**16.40.6.18** `status_t EDMA_DRV_SetChannelRequestAndTrigger ( uint8_t virtualChannel, uint8_t request, bool enableTrigger )`

Configures the DMA request for the eDMA channel.

Selects which DMA source is routed to a DMA channel. The DMA sources are defined in the file <MCU>\_↔ Features.h Configures the periodic trigger capability for the triggered DMA channel.

#### Parameters

<i>virtualChannel</i>	eDMA virtual channel number.
<i>request</i>	DMA request source.
<i>enableTrigger</i>	DMA channel periodic trigger.

#### Returns

STATUS\_SUCCESS or STATUS\_UNSUPPORTED.

Definition at line 970 of file edma\_driver.c.

**16.40.6.19** `void EDMA_DRV_SetDestAddr ( uint8_t virtualChannel, uint32_t address )`

Configures the destination address for the eDMA channel.

#### Parameters

<i>virtualChannel</i>	eDMA virtual channel number.
<i>address</i>	The pointer to the destination memory address.

Definition at line 1198 of file edma\_driver.c.

**16.40.6.20** `void EDMA_DRV_SetDestLastAddrAdjustment ( uint8_t virtualChannel, int32_t adjust )`

Configures the destination address last adjustment.

Adjustment value added to the destination address at the completion of the major iteration count. This value can be applied to restore the destination address to the initial value, or adjust the address to reference the next data structure.

#### Parameters

<i>virtualChannel</i>	eDMA virtual channel number.
<i>adjust</i>	Adjustment value.

Definition at line 1168 of file edma\_driver.c.

**16.40.6.21** void EDMA\_DRV\_SetDestOffset ( uint8\_t *virtualChannel*, int16\_t *offset* )

Configures the destination address signed offset for the eDMA channel.

Sign-extended offset applied to the current destination address to form the next-state value as each destination write is complete.

#### Parameters

<i>virtualChannel</i>	eDMA virtual channel number.
<i>offset</i>	signed-offset

Definition at line 1228 of file edma\_driver.c.

**16.40.6.22** void EDMA\_DRV\_SetDestWriteChunkSize ( uint8\_t *virtualChannel*, edma\_transfer\_size\_t *size* )

Configures the destination data chunk size (transferred in a write sequence).

Destination data write transfer size (1/2/4/16/32 bytes).

#### Parameters

<i>virtualChannel</i>	eDMA virtual channel number.
<i>size</i>	Destination transfer size.

Definition at line 1258 of file edma\_driver.c.

**16.40.6.23** void EDMA\_DRV\_SetMajorLoopIterationCount ( uint8\_t *virtualChannel*, uint32\_t *majorLoopCount* )

Configures the number of major loop iterations.

Sets the number of major loop iterations; each major loop iteration will be served upon a request for the current channel, transferring the data block configured for the minor loop (NBYTES).

#### Parameters

<i>virtualChannel</i>	eDMA virtual channel number.
<i>majorLoopCount</i>	Number of major loop iterations.

Definition at line 1318 of file edma\_driver.c.

**16.40.6.24** void EDMA\_DRV\_SetMinorLoopBlockSize ( uint8\_t *virtualChannel*, uint32\_t *nbytes* )

Configures the number of bytes to be transferred in each service request of the channel.

Sets the number of bytes to be transferred each time a request is received (one major loop iteration). This number needs to be a multiple of the source/destination transfer size, as the data block will be transferred within multiple read/write sequences (minor loops).

#### Parameters

<i>virtualChannel</i>	eDMA virtual channel number.
<i>nbytes</i>	Number of bytes to be transferred in each service request of the channel

Definition at line 1288 of file edma\_driver.c.

**16.40.6.25** void EDMA\_DRV\_SetScatterGatherLink ( uint8\_t *virtualChannel*, uint32\_t *nextTCDAddr* )

Configures the memory address of the next TCD, in scatter/gather mode.

This function configures the address of the next TCD to be loaded from memory, when scatter/gather feature is enabled. 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>virtualChannel</i>	eDMA virtual channel number.
<i>nextTCDAddr</i>	The address of the next TCD to be linked to this TCD.

Definition at line 1379 of file edma\_driver.c.

**16.40.6.26** void EDMA\_DRV\_SetSrcAddr ( uint8\_t *virtualChannel*, uint32\_t *address* )

Configures the source address for the eDMA channel.

#### Parameters

<i>virtualChannel</i>	eDMA virtual channel number.
<i>address</i>	The pointer to the source memory address.

Definition at line 1048 of file edma\_driver.c.

**16.40.6.27** void EDMA\_DRV\_SetSrcLastAddrAdjustment ( uint8\_t *virtualChannel*, int32\_t *adjust* )

Configures the source address last adjustment.

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>virtualChannel</i>	eDMA virtual channel number.
<i>adjust</i>	Adjustment value.

Definition at line 1138 of file edma\_driver.c.

**16.40.6.28** void EDMA\_DRV\_SetSrcOffset ( uint8\_t *virtualChannel*, int16\_t *offset* )

Configures the source address signed offset for the eDMA channel.

Sign-extended offset applied to the current source address to form the next-state value as each source read is complete.

#### Parameters

<i>virtualChannel</i>	eDMA virtual channel number.
<i>offset</i>	Signed-offset for source address.

Definition at line 1078 of file edma\_driver.c.

**16.40.6.29** void EDMA\_DRV\_SetSrcReadChunkSize ( uint8\_t *virtualChannel*, edma\_transfer\_size\_t *size* )

Configures the source data chunk size (transferred in a read sequence).

Source data read transfer size (1/2/4/16/32 bytes).

#### Parameters

<i>virtualChannel</i>	eDMA virtual channel number.
<i>size</i>	Source transfer size.

Definition at line 1108 of file edma\_driver.c.

**16.40.6.30** status\_t EDMA\_DRV\_StartChannel ( uint8\_t *virtualChannel* )

Starts an eDMA channel.

This function enables the eDMA channel DMA request.

**Parameters**

<i>virtualChannel</i>	eDMA virtual channel number.
-----------------------	------------------------------

**Returns**

STATUS\_ERROR or STATUS\_SUCCESS.

Definition at line 907 of file edma\_driver.c.

**16.40.6.31** `status_t EDMA_DRV_StopChannel ( uint8_t virtualChannel )`

Stops the eDMA channel.

This function disables the eDMA channel DMA request.

**Parameters**

<i>virtualChannel</i>	eDMA virtual channel number.
-----------------------	------------------------------

**Returns**

STATUS\_ERROR or STATUS\_SUCCESS.

Definition at line 938 of file edma\_driver.c.

**16.40.6.32** `void EDMA_DRV_TriggerSwRequest ( uint8_t virtualChannel )`

Triggers a sw request for the current channel.

This function starts a transfer using the current channel (sw request).

**Parameters**

<i>virtualChannel</i>	eDMA virtual channel number.
-----------------------	------------------------------

Definition at line 1516 of file edma\_driver.c.

## 16.41 EIM Driver

### 16.41.1 Detailed Description

Error Injection Module Peripheral Driver.

EIM PD provides a set of high-level APIs/services to configure the Error Injection Module (EIM) module.

#### Important Note:

1. Make sure that STACK memory is located in RAM different than where EIM will inject a non-correctable error.
2. For single bit error generation, flip only one bit out of DATA\_MASK or CHKBIT\_MASK bit-fields in EIM control registers.
3. For Double bit error generation, flip only two bits out of DATA\_MASK or CHKBIT\_MASK bit-fields in EIM control registers.
4. If more than 2 bits are flipped that there is no guarantee in design that what type of error get generated.
5. When generating double bit error or more than 2 bits error:
  - S32K11x: After injecting the error, the program jumps to HardFault\_Handler(). User needs to cancel the HardFault\_Handler() by disabling the EIM module inside the HardFault\_Handler() function. Example shown below: HardFault\_Handler() { EIM\_DRV\_Deinit(INST\_EIM1); }
  - S32Rx7x An uncorrectable ECC error occurs on an access generated by the DMA only. If a CPU access to the TCD causes an uncorrectable ECC error, that access will receive a bus error response.
6. When using double bit error generation on S32K11x, user needs to define one region called ram\_low then move the stack and m\_interrupts to that region, otherwise the module can't be enabled because the RAM ECC mechanism can only correct one single error.

#### Basic Operations of EIM

1. To initialize EIM, call [EIM\\_DRV\\_Init\(\)](#) with an user channel configuration array. In the following code, EIM is initialized with default settings (after reset) for check-bit mask and data mask and both channels is enabled.

```

1.1 With instance S32K14x
#define INST_EIM1 (0U)

#define EIM_CHANNEL_COUNT0 (2U)
/* Configuration structure array */
eim_user_channel_config_t userChannelConfigArr[] =
{
    /* Configuration channel 0 */
    {
        .channel = 0x0U,
        .checkBitMask = 0x00U,
        .dataMask = 0x00U,
        .enable = true
    },
    /* Configuration channel 1 */
    {
        .channel = 0x1U,
        .checkBitMask = 0x00U,
        .dataMask = 0x00U,
        .enable = true
    }
};
1.2 With instance S32K11x

```

```

#define INST_EIM1 (0U)

#define EIM_CHANNEL_COUNT0 (1U)
/* Configuration structure array */
eim_user_channel_config_t userChannelConfigArr[] =
{
    /* Configuration channel 0 */
    {
        .channel = 0x0U,
        .checkBitMask = 0x01U,
        .dataMask = 0x00U,
        .enable = true
    },
};
1.3 With instance S32Rx7x
#define INST_EIM1 (0U)

#define EIM_CHANNEL_COUNT0 (1U)
/* Configuration structure array */
eim_user_channel_config_t userChannelConfigArr[] =
{
    /* Configuration channel 0 */
    {
        .channel = 0x0U,
        .checkBitMask = 0x01U,
        .dataMask = 0x00U,
        .dataMask1= 0x00U,
        .enable = true
    },
};
/* Initialize the EIM instance 0 with configured channel number of 2 and userChannelConfigArr */
EIM_DRV_Init(INST_EIM1, EIM_CHANNEL_COUNT0 , userChannelConfigArr);

```

2. To get the default configuration (data mask, check-bit mask and enable status) of a channel in EIM, just call [EIM\\_DRV\\_GetDefaultConfig\(\)](#). Make sure that the operation is not execute in target RAM where EIM inject the error

```

eim_user_channel_config_t channelConfig;

/* Get default configuration of EIM channel 1*/
EIM_DRV_GetDefaultConfig(1U, &channelConfig);

```

3. To de-initialize EIM, just call the [EIM\\_DRV\\_Deinit\(\)](#) function. This function sets all registers to reset values and disables EIM.

```

/* De-initializes the EIM module */
EIM_DRV_Deinit(INST_EIM1);

```

## Data Structures

- struct [eim\\_user\\_channel\\_config\\_t](#)  
*EIM channel configuration structure. [More...](#)*

## Macros

- #define [EIM\\_CHECKBITMASK\\_DEFAULT](#) (0x01U)  
*The value default of EIM check-bit mask.*
- #define [EIM\\_DATAMASK\\_DEFAULT](#) (0x00U)  
*The value default of EIM data mask.*

## EIM Driver API

- void [EIM\\_DRV\\_Init](#) (uint32\_t instance, uint8\_t channelCnt, const [eim\\_user\\_channel\\_config\\_t](#) \*channelConfigArr)  
*Initializes the EIM module.*

- void `EIM_DRV_Deinit` (uint32\_t instance)  
*De-initializes the EIM module.*
- void `EIM_DRV_ConfigChannel` (uint32\_t instance, const `eim_user_channel_config_t` \*userChannelConfig)  
*Configures the EIM channel.*
- void `EIM_DRV_GetChannelConfig` (uint32\_t instance, uint8\_t channel, `eim_user_channel_config_t` \*channelConfig)  
*Gets the EIM channel configuration.*
- void `EIM_DRV_GetDefaultConfig` (uint8\_t channel, `eim_user_channel_config_t` \*channelConfig)  
*Gets the EIM channel configuration default.*

## 16.41.2 Data Structure Documentation

### 16.41.2.1 struct `eim_user_channel_config_t`

EIM channel configuration structure.

This structure holds the configuration settings for the EIM channel Implements : `eim_user_channel_config_t_Class`  
Definition at line 55 of file `eim_driver.h`.

#### Data Fields

- uint8\_t `channel`
- uint8\_t `checkBitMask`
- uint32\_t `dataMask`
- bool `enable`

#### Field Documentation

##### 16.41.2.1.1 uint8\_t channel

EIM channel number

Definition at line 57 of file `eim_driver.h`.

##### 16.41.2.1.2 uint8\_t checkBitMask

Specifies whether the corresponding bit of the check-bit bus from the target RAM should be inverted or remain unmodified

Definition at line 58 of file `eim_driver.h`.

##### 16.41.2.1.3 uint32\_t dataMask

Specifies whether the corresponding bit of the read data bus from the target RAM should be inverted or remain unmodified

Definition at line 60 of file `eim_driver.h`.

##### 16.41.2.1.4 bool enable

true : EIM channel operation is enabled false : EIM channel operation is disabled

Definition at line 66 of file `eim_driver.h`.

## 16.41.3 Macro Definition Documentation

### 16.41.3.1 #define `EIM_CHECKBITMASK_DEFAULT` (0x01U)

The value default of EIM check-bit mask.

Definition at line 45 of file `eim_driver.h`.

### 16.41.3.2 #define EIM\_DATAMASK\_DEFAULT (0x00U)

The value default of EIM data mask.

Definition at line 47 of file eim\_driver.h.

## 16.41.4 Function Documentation

### 16.41.4.1 void EIM\_DRV\_ConfigChannel ( uint32\_t instance, const eim\_user\_channel\_config\_t \* userChannelConfig )

Configures the EIM channel.

This function configures check-bit mask, data mask and operation status(enable/disable) for EIM channel. The EIM channel configuration structure shall be passed as arguments.

This is an example demonstrating how to define a EIM channel configuration structure:

```
1 eim_user_channel_config_t eimTestInit = {
2     .channel = 0x1U,
3     .checkBitMask = 0x25U,
4     .dataMask = 0x11101100U,
5     .enable = true
6 };
```

#### Parameters

in	<i>instance</i>	EIM module instance number
in	<i>userChannelConfig</i>	Pointer to EIM channel configuration structure

Definition at line 115 of file eim\_driver.c.

### 16.41.4.2 void EIM\_DRV\_Deinit ( uint32\_t instance )

De-initializes the EIM module.

This function sets all registers to reset value and disables EIM module. In order to use the EIM module again, EIM\_DRV\_Init must be called.

#### Parameters

in	<i>instance</i>	EIM module instance number
----	-----------------	----------------------------

Definition at line 92 of file eim\_driver.c.

### 16.41.4.3 void EIM\_DRV\_GetChannelConfig ( uint32\_t instance, uint8\_t channel, eim\_user\_channel\_config\_t \* channelConfig )

Gets the EIM channel configuration.

This function gets check bit mask, data mask and operation status of EIM channel.

#### Parameters

in	<i>instance</i>	EIM module instance number
in	<i>channel</i>	EIM channel number
out	<i>channelConfig</i>	Pointer to EIM channel configuration structure

Definition at line 145 of file eim\_driver.c.

### 16.41.4.4 void EIM\_DRV\_GetDefaultConfig ( uint8\_t channel, eim\_user\_channel\_config\_t \* channelConfig )

Gets the EIM channel configuration default.

This function gets check bit mask, data mask and operation status default of EIM channel.

## Parameters

in	<i>channel</i>	EIM channel number
out	<i>channelConfig</i>	Pointer to EIM channel configuration structure default

Definition at line 176 of file eim\_driver.c.

**16.41.4.5** void EIM\_DRV\_Init ( uint32\_t *instance*, uint8\_t *channelCnt*, const eim\_user\_channel\_config\_t \* *channelConfigArr* )

Initializes the EIM module.

This function configures for EIM channels. The EIM channel configuration structure array and number of configured channels shall be passed as arguments. This function should be called before calling any other EIM driver function.

This is an example demonstrating how to define a EIM channel configuration structure array:

```

1 eim_user_channel_config_t channelConfigArr[] =
2 {
3 {
4 .channel = 0x0U,
5 .checkBitMask = 0x12U,
6 .dataMask = 0x01234567U,
7 .enable = true
8 },
9 {
10 .channel = 0x1U,
11 .checkBitMask = 0x22U,
12 .dataMask = 0x01234444U,
13 .enable = false
14 }
15 };

```

## Parameters

in	<i>instance</i>	EIM module instance number.
in	<i>channelCnt</i>	Number of configured channels
in	<i>channelConfigArr</i>	EIM channel configuration structure array

Definition at line 62 of file eim\_driver.c.

## 16.42 ERM Driver

### 16.42.1 Detailed Description

Error Reporting Module Peripheral Driver.

This section describes the programming interface of the ERM driver.

### 16.42.2 ERM Driver Initialization

In order to be able to use the error reporting in your application, the first thing to do is initializing it with user configuration input. This is done by calling the **ERM\_DRV\_Init** function. Note that: channelCnt takes values between 1 and the maximum channel count supported by the hardware.

### 16.42.3 ERM Driver Operation

After ERM initialization, the **ERM\_DRV\_SetInterruptConfig()** shall be used to set interrupt notification based on interrupt notification configuration.

The **ERM\_DRV\_GetInterruptConfig()** shall be used to get the current interrupt configuration of the available events (which interrupts are enabled/disabled).

The **ERM\_DRV\_GetErrorDetail()** shall be used to get the address of the last ECC event in Memory n and ECC event.

The **ERM\_DRV\_ClearEvent()** shall be used to clear both the record of an event and the corresponding interrupt notification.

This is example code to configure the ERM driver:

```
/* Device instance number */
#define INST_ERM1 (0U)

/* 1.1 With instance for S32K14x: */
/* The number of configured channel(s) */
#define ERM_NUM_OF_CFG_CHANNEL (2U)

/* Interrupt configuration 0 */
const erm_interrupt_config_t erm1_interrupt0 =
{
    .enableSingleCorrection = false,
    .enableNonCorrectable   = true
};

/* Interrupt configuration 1 */
const erm_interrupt_config_t erm1_interrupt1 =
{
    .enableSingleCorrection = true,
    .enableNonCorrectable   = true
};

/* User configuration */
const erm_user_config_t erm1_InitConfig[] =
{
    /* Channel 0U */
    {
        .channel      = 0U,
        .interruptCfg = &erm1_interrupt0
    },

    /* Channel 1U */
    {
        .channel      = 1U,
        .interruptCfg = &erm1_interrupt1
    }
};

/* 1.2 With instance for S32K11x: */
/* The number of configured channel(s) */
#define ERM_NUM_OF_CFG_CHANNEL (1U)
```



```

/* Interrupt configuration 0 */
const erm_interrupt_config_t erm1_interrupt0 =
{
    .enableSingleCorrection = false,
    .enableNonCorrectable   = true
};

/* User configuration */
const erm_user_config_t erm1_InitConfig[] =
{
    /* Channel 0U */
    {
        .channel      = 0U,
        .interruptCfg = &erm1_interrupt0
    }
};

int main()
{
    /* Initializes the ERM module */
    ERM_DRV_Init(INST_ERM1, ERM_NUM_OF_CFG_CHANNEL, erm1_InitConfig);
    ...
    /* De-Initializes the ERM module */
    ERM_DRV_Deinit(INST_ERM1);
    ...
    return 0;
}

/* Interrupt handler */
/* Interrupt handler for single bit */
void ERM_single_fault_IRQHandler()
{
    /* Clears the event for channel 1 */
    ERM_DRV_ClearEvent(INST_ERM1, 1U, ERM_EVENT_SINGLE_BIT);
    ...
}

/* Interrupt handler for non correctable */
void ERM_double_fault_IRQHandler()
{
    /* Clears the event for channel 0 */
    ERM_DRV_ClearEvent(INST_ERM1, 0U,
        ERM_EVENT_NON_CORRECTABLE);
    /* Clears the event for channel 1 */
    ERM_DRV_ClearEvent(INST_ERM1, 1U,
        ERM_EVENT_NON_CORRECTABLE);
    ...
}

```

## Data Structures

- struct [erm\\_interrupt\\_config\\_t](#)  
ERM interrupt notification configuration structure Implements : [erm\\_interrupt\\_config\\_t\\_Class](#). [More...](#)
- struct [erm\\_user\\_config\\_t](#)  
ERM user configuration structure Implements : [erm\\_user\\_config\\_t\\_Class](#). [More...](#)

## Enumerations

- enum [erm\\_ecc\\_event\\_t](#) { [ERM\\_EVENT\\_NONE](#) = 0U, [ERM\\_EVENT\\_SINGLE\\_BIT](#) = 1U, [ERM\\_EVENT\\_NON\\_CORRECTABLE](#) = 2U }
- ERM types of ECC events Implements : [erm\\_ecc\\_event\\_t\\_Class](#).

## ERM DRIVER API

- void [ERM\\_DRV\\_Init](#) (uint32\_t instance, uint8\_t channelCnt, const [erm\\_user\\_config\\_t](#) \*userConfigArr)  
Initializes the ERM module.
- void [ERM\\_DRV\\_Deinit](#) (uint32\_t instance)  
Sets the default configuration.
- void [ERM\\_DRV\\_SetInterruptConfig](#) (uint32\_t instance, uint8\_t channel, [erm\\_interrupt\\_config\\_t](#) interruptCfg)

*Sets interrupt notification.*

- void [ERM\\_DRV\\_GetInterruptConfig](#) (uint32\_t instance, uint8\_t channel, [erm\\_interrupt\\_config\\_t](#) \*const interruptPtr)

*Gets interrupt notification.*

- void [ERM\\_DRV\\_ClearEvent](#) (uint32\_t instance, uint8\_t channel, [erm\\_ecc\\_event\\_t](#) eccEvent)

*Clears error event and the corresponding interrupt notification.*

- [erm\\_ecc\\_event\\_t](#) [ERM\\_DRV\\_GetErrorDetail](#) (uint32\_t instance, uint8\_t channel, uint32\_t \*addressPtr)

*Gets the address of the last ECC event in Memory n and ECC event.*

#### 16.42.4 Data Structure Documentation

##### 16.42.4.1 struct [erm\\_interrupt\\_config\\_t](#)

ERM interrupt notification configuration structure Implements : [erm\\_interrupt\\_config\\_t\\_Class](#).

Definition at line 53 of file [erm\\_driver.h](#).

##### Data Fields

- bool [enableSingleCorrection](#)
- bool [enableNonCorrectable](#)

##### Field Documentation

###### 16.42.4.1.1 bool [enableNonCorrectable](#)

Enable Non-Correctable Interrupt Notification

Definition at line 56 of file [erm\\_driver.h](#).

###### 16.42.4.1.2 bool [enableSingleCorrection](#)

Enable Single Correction Interrupt Notification

Definition at line 55 of file [erm\\_driver.h](#).

##### 16.42.4.2 struct [erm\\_user\\_config\\_t](#)

ERM user configuration structure Implements : [erm\\_user\\_config\\_t\\_Class](#).

Definition at line 63 of file [erm\\_driver.h](#).

##### Data Fields

- uint8\_t [channel](#)
- const [erm\\_interrupt\\_config\\_t](#) \* [interruptCfg](#)

##### Field Documentation

###### 16.42.4.2.1 uint8\_t [channel](#)

The channel assignments

Definition at line 65 of file [erm\\_driver.h](#).

###### 16.42.4.2.2 const [erm\\_interrupt\\_config\\_t](#)\* [interruptCfg](#)

Interrupt configuration

Definition at line 66 of file [erm\\_driver.h](#).

## 16.42.5 Enumeration Type Documentation

## 16.42.5.1 enum erm\_ecc\_event\_t

ERM types of ECC events Implements : erm\_ecc\_event\_t\_Class.

## Enumerator

**ERM\_EVENT\_NONE** None events

**ERM\_EVENT\_SINGLE\_BIT** Single-bit correction ECC events

**ERM\_EVENT\_NON\_CORRECTABLE** Non-correctable ECC events

Definition at line 42 of file erm\_driver.h.

## 16.42.6 Function Documentation

## 16.42.6.1 void ERM\_DRV\_ClearEvent ( uint32\_t instance, uint8\_t channel, erm\_ecc\_event\_t eccEvent )

Clears error event and the corresponding interrupt notification.

This function clears the record of an event. If the corresponding interrupt is enabled, the interrupt notification will be cleared

## Parameters

in	instance	The ERM instance number
in	channel	The configured memory channel
in	eccEvent	The types of ECC events

Definition at line 142 of file erm\_driver.c.

## 16.42.6.2 void ERM\_DRV\_Deinit ( uint32\_t instance )

Sets the default configuration.

This function sets the default configuration

## Parameters

in	instance	The ERM instance number
----	----------	-------------------------

Definition at line 82 of file erm\_driver.c.

## 16.42.6.3 erm\_ecc\_event\_t ERM\_DRV\_GetErrorDetail ( uint32\_t instance, uint8\_t channel, uint32\_t \* addressPtr )

Gets the address of the last ECC event in Memory n and ECC event.

This function gets the address of the last ECC event in Memory n and the types of the event

## Parameters

in	instance	The ERM instance number
in	channel	The examined memory channel
out	addressPtr	The pointer to address of the last ECC event in Memory n with ECC event

## Returns

The last occurred ECC event

Definition at line 174 of file erm\_driver.c.

**16.42.6.4 void ERM\_DRV\_GetInterruptConfig ( uint32\_t *instance*, uint8\_t *channel*, erm\_interrupt\_config\_t \*const *interruptPtr* )**

Gets interrupt notification.

This function gets the current interrupt configuration of the available events (which interrupts are enabled/disabled)

**Parameters**

in	<i>instance</i>	The ERM instance number
in	<i>channel</i>	The examined memory channel
out	<i>interruptPtr</i>	The pointer to the ERM interrupt configuration structure

Definition at line 120 of file erm\_driver.c.

**16.42.6.5 void ERM\_DRV\_Init ( uint32\_t *instance*, uint8\_t *channelCnt*, const erm\_user\_config\_t \* *userConfigArr* )**

Initializes the ERM module.

This function initializes ERM driver based on user configuration input, channelCnt takes values between 1 and the maximum channel count supported by the hardware

**Parameters**

in	<i>instance</i>	The ERM instance number
in	<i>channelCnt</i>	The number of channels
in	<i>userConfigArr</i>	The pointer to the array of ERM user configure structure

Definition at line 54 of file erm\_driver.c.

**16.42.6.6 void ERM\_DRV\_SetInterruptConfig ( uint32\_t *instance*, uint8\_t *channel*, erm\_interrupt\_config\_t *interruptCfg* )**

Sets interrupt notification.

This function sets interrupt notification based on interrupt notification configuration input

**Parameters**

in	<i>instance</i>	The ERM instance number
in	<i>channel</i>	The configured memory channel
in	<i>interruptCfg</i>	The ERM interrupt configuration structure

Definition at line 99 of file erm\_driver.c.

## 16.43 EWM Driver

### 16.43.1 Detailed Description

External Watchdog Monitor Peripheral Driver.

#### Hardware background

Features:

- Independent LPO clock source
- Programmable time-out period specified in terms of number of EWM LPO clock cycles.
- Windowed refresh option
  - Provides robust check that program flow is faster than expected.
  - Programmable window.
  - Refresh outside window leads to assertion of EWM\_out.
- Robust refresh mechanism
  - Write values of **0xB4** and **0x2C** to EWM Refresh Register within 15 (**EWM\_service\_time**) peripheral bus clock cycles.
- One output port, **EWM\_out**, when asserted is used to reset or place the external circuit into safe mode
- One Input port, **EWM\_in**, allows an external circuit to control the **EWM\_out** signal.

**The EWM can be initialized only once as all the configuration registers are write once per reset**

#### Integration guideline

##### Compilation units

The following files need to be compiled in the project:

```
* ${S32SDK_PATH}\platform\drivers\src\ewm\ewm_driver.c
*
```

##### Include path

The following paths need to be added to the include path of the toolchain:

```
* ${S32SDK_PATH}\platform\drivers\inc\
*
```

##### Compile symbols

No special symbols are required for this component

##### Dependencies

No special dependencies are required for this component

##### Clocking and pin configuration

The EWM Driver does not handle clock setup (from PCC) or any kind of pin configuration (done by PORT module). This is handled by the Clock Manager and PORT module, respectively. The driver assumes that correct clock configurations have been made, so it is the user's responsibility to set up clocking and pin configurations correctly.

## Interrupts

The EWM module can generate interrupts, if enabled on `EWM_DRV_Init()` but they are not handled by the driver. The EWM shares the interrupt vector with the Watchdog Timer. The following code snippet is an example of how enable the interrupt and assign a handler:

```
/* EWM and watchdog interrupt service routine */
void EWM_Watchdog_ISR()
{
    /* Do something(e.g perform a clean reset) */
    ...
}

int main()
{
    /* Init clocks, pins, other modules */
    ...
    /* Install interrupt handler for EWM and Watchdog */
    INT_SYS_InstallHandler(WDOG_EWM_IRQn, &EWM_Watchdog_ISR, (
        isr_t *)0);
    /* Enable the interrupt */
    INT_SYS_EnableIRQ(WDOG_EWM_IRQn);

    /* Init EWM */
    ...
    /* Infinite loop*/
    while(1)
    {
        /* Do something until the counter needs to be refreshed */
        ...
        /* Refresh the counter */
        EWM_DRV_Refresh(EWM_INSTANCE);
    }
}
```

## Using the EWM driver in your application

```
/* Declare the EWM instance you want to use */
#define EWM_INSTANCE    OUL

int main()
{
    /* Declare the EWM configuration structure */
    ewm_init_config_t ewmConfig;
    /* Variable where to store the init status */
    status_t ewmStatus;
    /* Init clocks, pins, other modules */
    ...

    /* Get the default configuration values */
    EWM_DRV_GetDefaultConfig(&ewmConfig);
    /* Init the module instance */
    ewmStatus = EWM_DRV_Init(EWM_INSTANCE, &ewmConfig);

    /* Infinite loop*/
    while(1)
    {
        /* Do something until the counter needs to be refreshed */
        ...
        /* Refresh the counter */
        EWM_DRV_Refresh(EWM_INSTANCE);
    }
}
```

## Data Structures

- struct `ewm_init_config_t`

*EWM configuration structure This structure is used to configure the EWM prescaler, window, interrupt and input pin.  
More...*

## Enumerations

- enum `ewm_in_assert_logic_t` { `EWM_IN_ASSERT_DISABLED` = 0x00U, `EWM_IN_ASSERT_ON_LOGIC_ZERO` = 0x01U, `EWM_IN_ASSERT_ON_LOGIC_ONE` = 0x02U }

*EWM input pin configuration Configures if the input pin is enabled and when is asserted Implements : ewm\_in\_assert\_logic\_t Class.*

## EWM Driver API

- status\_t [EWM\\_DRV\\_Init](#) (uint32\_t instance, const [ewm\\_init\\_config\\_t](#) \*config)  
*Init EWM. This method initializes EWM instance to the configuration from the passed structure. The user must make sure that the clock is enabled. This is the only method needed to be called to start the module.*
- void [EWM\\_DRV\\_GetDefaultConfig](#) ([ewm\\_init\\_config\\_t](#) \*config)  
*Init configuration structure to default values.*
- void [EWM\\_DRV\\_Refresh](#) (uint32\_t instance)  
*Refresh EWM. This method needs to be called within the window period specified by the Compare Low and Compare High registers.*
- [ewm\\_in\\_assert\\_logic\\_t](#) [EWM\\_DRV\\_GetInputPinAssertLogic](#) (uint32\_t instance)  
*Get the Input pin assert logic.*

## 16.43.2 Data Structure Documentation

16.43.2.1 struct [ewm\\_init\\_config\\_t](#)

EWM configuration structure This structure is used to configure the EWM prescaler, window, interrupt and input pin.

Implements : [ewm\\_init\\_config\\_t\\_Class](#)

Definition at line 54 of file [ewm\\_driver.h](#).

## Data Fields

- [ewm\\_in\\_assert\\_logic\\_t](#) [assertLogic](#)
- bool [interruptEnable](#)
- uint8\_t [prescaler](#)
- uint8\_t [compareLow](#)
- uint8\_t [compareHigh](#)

## Field Documentation

16.43.2.1.1 [ewm\\_in\\_assert\\_logic\\_t](#) [assertLogic](#)

Assert logic for EWM input pin

Definition at line 56 of file [ewm\\_driver.h](#).

16.43.2.1.2 uint8\_t [compareHigh](#)

Compare high value

Definition at line 60 of file [ewm\\_driver.h](#).

16.43.2.1.3 uint8\_t [compareLow](#)

Compare low value

Definition at line 59 of file [ewm\\_driver.h](#).

16.43.2.1.4 bool [interruptEnable](#)

Enable EWM interrupt

Definition at line 57 of file [ewm\\_driver.h](#).

16.43.2.1.5 uint8\_t [prescaler](#)

EWM clock prescaler

Definition at line 58 of file [ewm\\_driver.h](#).

### 16.43.3 Enumeration Type Documentation

#### 16.43.3.1 enum ewm\_in\_assert\_logic\_t

EWM input pin configuration Configures if the input pin is enabled and when is asserted Implements : ewm\_in\_assert\_logic\_t\_Class.

#### Enumerator

- EWM\_IN\_ASSERT\_DISABLED** Input pin disabled  
**EWM\_IN\_ASSERT\_ON\_LOGIC\_ZERO** Input pin asserts EWM when on logic 0  
**EWM\_IN\_ASSERT\_ON\_LOGIC\_ONE** Input pin asserts EWM when on logic 1

Definition at line 40 of file ewm\_driver.h.

### 16.43.4 Function Documentation

#### 16.43.4.1 void EWM\_DRV\_GetDefaultConfig ( ewm\_init\_config\_t \* config )

Init configuration structure to default values.

#### Parameters

out	config	Pointer to the configuration structure to initialize
-----	--------	--

#### Returns

None

Definition at line 128 of file ewm\_driver.c.

#### 16.43.4.2 ewm\_in\_assert\_logic\_t EWM\_DRV\_GetInputPinAssertLogic ( uint32\_t instance )

Get the Input pin assert logic.

#### Parameters

in	instance	EWM instance number
----	----------	---------------------

#### Returns

The input pin assert logic:

- EWM\_IN\_ASSERT\_DISABLED - EWM in disabled
- EWM\_IN\_ASSERT\_ON\_LOGIC\_ZERO - EWM is asserted when EWM\_in is logic 0
- EWM\_IN\_ASSERT\_ON\_LOGIC\_ONE - EWM is asserted when EWM\_in is logic 1

Definition at line 172 of file ewm\_driver.c.

#### 16.43.4.3 status\_t EWM\_DRV\_Init ( uint32\_t instance, const ewm\_init\_config\_t \* config )

Init EWM. This method initializes EWM instance to the configuration from the passed structure. The user must make sure that the clock is enabled. This is the only method needed to be called to start the module.

Example configuration structure:

```
1 ewm_init_config_t ewmUserCfg = {
2     .assertLogic      = EWM_IN_ASSERT_ON_LOGIC_ZERO,
3     .interruptEnable   = true,
4     .prescaler         = 128,
5     .compareLow        = 0,
6     .compareHigh       = 254
7 };
```



This configuration will enable the peripheral, with input pin configured to assert on logic low, interrupt enabled, prescaler 128 and maximum refresh window.

The EWM can be initialized only once per CPU reset as the registers are write once.

#### Parameters

<i>in</i>	<i>instance</i>	EWM instance number
<i>in</i>	<i>config</i>	Pointer to the module configuration structure.

#### Returns

`status_t` Will return the status of the operation:

- `STATUS_SUCCESS` if the operation is successful
- `STATUS_ERROR` if the windows values are not correct or if the instance is already enabled

Definition at line 60 of file `ewm_driver.c`.

#### 16.43.4.4 void EWM\_DRV\_Refresh ( uint32\_t *instance* )

Refresh EWM. This method needs to be called within the window period specified by the Compare Low and Compare High registers.

#### Parameters

<i>in</i>	<i>instance</i>	EWM instance number
-----------	-----------------	---------------------

#### Returns

None

Definition at line 150 of file `ewm_driver.c`.

## 16.44 Enhanced Direct Memory Access (eDMA)

### 16.44.1 Detailed Description

The S32 SDK provides Peripheral Driver for the Enhanced Direct Memory Access (eDMA) module. The direct memory access engine features are used for performing complex data transfers with minimal intervention from the host processor. These sections describe the S32 SDK software modules API that can be used for initializing, configuring and triggering eDMA transfers.

#### Modules

- [EDMA Driver](#)

*This module covers the functionality of the Enhanced Direct Memory Access (eDMA) peripheral driver.*

## 16.45 Error Injection Module (EIM)

### 16.45.1 Detailed Description

The S32 SDK provides Peripheral Drivers for the Error Injection Module (EIM) of S32 MCU.

The Error Injection Module (EIM) is mainly used for diagnostic purposes. It provides a method for diagnostic coverage of the peripheral memories and offers support for inducing single-bit and multi-bit inversions on read data when accessing peripheral RAMs.

Injecting faults on memory accesses can be used to exercise the SEC-DED ECC function of the related system. Each EIM channel *n* corresponds to a source of potential memory error events.

The following table shows the channel assignments of the module:

EIM channel <i>n</i>	S32K14x	S32↔ K14xW	S32K11x	S32Rx7x	MP↔ C5746R	MP↔ C5777C	S32V23x
0	SRAM_L	SRAM_L	SRAM_U	DMA TCD RAM	DMA TCD RAM	PRAMC↔ _0	Cortex-M4 TCM upper (bits31-0)
1	SRAM_U	SRAM_U	Reserved	Reserved	Reserved	PRAMC↔ _1	Cortex-M4 TCM upper (bits63-32)
2	Reserved	Reserved	Reserved	Reserved	Reserved	FEC MIB	Cortex-M4 TCM lower (bits31-0)
3	Reserved	Reserved	Reserved	Reserved	Reserved	FEC RIF	Cortex-M4 TCM lower (bits63-32)
4	Reserved	Reserved	Reserved	Reserved	Reserved	eDMA_0 TCD RAM	Cortex-M4 Code Cache Tag Way0
5	Reserved	Reserved	Reserved	Reserved	Reserved	eDMA_1 TCD RAM	Cortex-M4 Code Cache Tag Way1
6	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Cortex-M4 Code Cache Data Way0
7	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Cortex-M4 Code Cache Data Way1
8	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Cortex-M4 System Cache Tag Way0
9	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Cortex-M4 System Cache Tag Way1

10	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Cortex-M4 System Cache Data Way0
11	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Cortex-M4 System Cache Data Way1
12	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	DMA TCD RAM

/\*!

#### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```

${S32SDK_PATH}\platform\drivers\src\eim\eim_driver.c
${S32SDK_PATH}\platform\drivers\src\eim\eim_hw_access.c

```

#### Include path

The following paths need to be added to the include path of the toolchain:

```

${S32SDK_PATH}\platform\drivers\inc\

```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

No special dependencies are required for this component

#### Modules

- [EIM Driver](#)

*Error Injection Module Peripheral Driver.*

*EIM PD provides a set of high-level APIs/services to configure the Error Injection Module (EIM) module.*

## 16.46 Error Reporting Module (ERM)

### 16.46.1 Detailed Description

The S32 SDK provides Peripheral Drivers for the Error Reporting Module (ERM) module of S32 SDK devices.

The Error Reporting Module (ERM) provides information and optional interrupt notification on memory errors events associated with ECC (Error Correction Code).

The ERM includes the following features:

- Capture of address information on single-bit correction and non-correctable ECC events.
- Optional interrupt notification on captured ECC events.
- Support for ECC event capturing for memory sources, with individual reporting fields and interrupt configuration per memory channel.

Each ERM channel *n* corresponds to a source of potential memory error events. The following table shows the channel assignments:

ERM channel <i>n</i>	S32K14x	S32K14xW	S32K11x	MPC5777C	S32V23x
0	SRAM_L	SRAM_L	SRAM_U	PRAMC_0	Cortex-M4 TCM upper
1	SRAM_U	SRAM_U	Reserved	PRAMC_1	Cortex-M4 TCM lower
2	Reserved	Reserved	Reserved	eDMA_0 TCD RAM	Cortex-M4 Code Cache Tag
3	Reserved	Reserved	Reserved	eDMA_1 TCD RAM	Cortex-M4 Code Cache Data
4	Reserved	Reserved	Reserved	FEC MIB	Cortex-M4 System Cache Tag
5	Reserved	Reserved	Reserved	FEC RIF	Cortex-M4 System Cache Data
6	Reserved	Reserved	Reserved	PFLASH port 0	DMA TCD RAM
7	Reserved	Reserved	Reserved	PFLASH port 1	Reserved
8	Reserved	Reserved	Reserved	AIPS_0	Reserved
9	Reserved	Reserved	Reserved	AIPS_1	Reserved
10	Reserved	Reserved	Reserved	FEC e2eECC	Reserved
11	Reserved	Reserved	Reserved	CSE	Reserved
12	Reserved	Reserved	Reserved	SIPI	Reserved
13	Reserved	Reserved	Reserved	Core0 instruction	Reserved
14	Reserved	Reserved	Reserved	Core0 data	Reserved
15	Reserved	Reserved	Reserved	Core1 instruction	Reserved
16	Reserved	Reserved	Reserved	Core1 data	Reserved
17	Reserved	Reserved	Reserved	eDMA_0 e2eECC	Reserved

18	Reserved	Reserved	Reserved	eDMA_1 e2eECC	Reserved
19	Reserved	Reserved	Reserved	EBI e2eECC	Reserved

#### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```

${S32SDK_PATH}\platform\drivers\src\erm\erm_driver.c
${S32SDK_PATH}\platform\drivers\src\erm\erm_hw_access.c

```

#### Include path

The following paths need to be added to the include path of the toolchain:

```

${S32SDK_PATH}\platform\drivers\inc\

```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

No special dependencies are required for this component

#### Modules

- [ERM Driver](#)  
*Error Reporting Module Peripheral Driver.*

## 16.47 External Watchdog Monitor (EWM)

### 16.47.1 Detailed Description

The S32 SDK provides the Peripheral Drivers for the External Watchdog Monitor (EWM) module of S32K devices.

For safety, a redundant watchdog system, External Watchdog Monitor (EWM), is designed to monitor external circuits, as well as the MCU software flow. This provides a back-up mechanism to the internal watchdog that resets the MCU's CPU and peripherals.

The EWM differs from the internal watchdog in that it does not reset the MCU's CPU and peripherals. The EWM if allowed to time-out, provides an independent EWM\_out pin that when asserted resets or places an external circuit into a safe mode. The CPU resets the EWM counter that is logically ANDed with an external digital input pin. This pin allows an external circuit to influence the reset\_out signal.

### Modules

- [EWM Driver](#)

*External Watchdog Monitor Peripheral Driver.*

## 16.48 Flash Memory (Flash)

### 16.48.1 Detailed Description

This section describes the programming interface of the Flash Peripheral Driver.

#### Data Structures

- struct `flash_user_config_t`  
*Flash User Configuration Structure. [More...](#)*
- struct `flash_ssd_config_t`  
*Flash SSD Configuration Structure. [More...](#)*
- struct `flash_eeprom_status_t`  
*EEPROM status structure. [More...](#)*

#### Macros

- #define `CLEAR_FTFx_FSTAT_ERROR_BITS` `FTFx_FSTAT = (uint8_t)(FTFx_FSTAT_FPVIOL_MASK | FTFx_FSTAT_ACCERR_MASK | FTFx_FSTAT_RDCOLERR_MASK)`
- #define `FTFx_WORD_SIZE` `0x0002U`
- #define `FTFx_LONGWORD_SIZE` `0x0004U`
- #define `FTFx_PHRASE_SIZE` `0x0008U`
- #define `FTFx_DPHRASE_SIZE` `0x0010U`
- #define `FTFx_RSRC_CODE_REG` `FTFx_FCCOB8`
- #define `FTFx_VERIFY_BLOCK` `0x00U`
- #define `FTFx_VERIFY_SECTION` `0x01U`
- #define `FTFx_PROGRAM_CHECK` `0x02U`
- #define `FTFx_READ_RESOURCE` `0x03U`
- #define `FTFx_PROGRAM_LONGWORD` `0x06U`
- #define `FTFx_PROGRAM_PHRASE` `0x07U`
- #define `FTFx_ERASE_BLOCK` `0x08U`
- #define `FTFx_ERASE_SECTOR` `0x09U`
- #define `FTFx_PROGRAM_SECTION` `0x0BU`
- #define `FTFx_VERIFY_ALL_BLOCK` `0x40U`
- #define `FTFx_READ_ONCE` `0x41U`
- #define `FTFx_PROGRAM_ONCE` `0x43U`
- #define `FTFx_ERASE_ALL_BLOCK` `0x44U`
- #define `FTFx_SECURITY_BY_PASS` `0x45U`
- #define `FTFx_PFLASH_SWAP` `0x46U`
- #define `FTFx_ERASE_ALL_BLOCK_UNSECURE` `0x49U`
- #define `FTFx_PROGRAM_PARTITION` `0x80U`
- #define `FTFx_SET_EERAM` `0x81U`
- #define `RESUME_WAIT_CNT` `0x20U`  
*Resume wait count used in FLASH\_DRV\_EraseResume function.*
- #define `SUSPEND_WAIT_CNT` `0x40U`  
*Suspend wait count used in FLASH\_DRV\_EraseSuspend function.*
- #define `DFLASH_IFR_READRESOURCE_ADDRESS` `0x8000FCU`
- #define `GET_BIT_0_7(value)` `((uint8_t)((((uint32_t)(value)) & 0xFFU))`
- #define `GET_BIT_8_15(value)` `((uint8_t)((((uint32_t)(value)) >> 8) & 0xFFU))`
- #define `GET_BIT_16_23(value)` `((uint8_t)((((uint32_t)(value)) >> 16) & 0xFFU))`
- #define `GET_BIT_24_31(value)` `((uint8_t)((((uint32_t)(value)) >> 24))`
- #define `FLASH_SECURITY_STATE_KEYEN` `0x80U`
- #define `FLASH_SECURITY_STATE_UNSECURED` `0x02U`



- #define `CSE_KEY_SIZE_CODE_MAX` 0x03U
- #define `FTFx_FSTAT_ERROR_BITS` (FTFx\_FSTAT & (FTFx\_FSTAT\_MGSTAT0\_MASK | FTFx\_FSTAT\_↵\_FPVIOL\_MASK | FTFx\_FSTAT\_ACCERR\_MASK | FTFx\_FSTAT\_RDCOLERR\_MASK))
- #define `FLASH_CALLBACK_CS` 0x0AU  
*Callback period count for FlashCheckSum.*

### Typedefs

- typedef void(\* `flash_callback_t`) (void)  
*Call back function pointer data type.*

### Enumerations

- enum `flash_flexRam_function_control_code_t` {  
`EEE_ENABLE` = 0x00U, `EEE_QUICK_WRITE` = 0x55U, `EEE_STATUS_QUERY` = 0x77U, `EEE_COMPL↵ETE_INTERRUPT_QUICK_WRITE` = 0xAAU,  
`EEE_DISABLE` = 0xFFU }  
*FlexRAM Function control Code.*

### Variables

- uint32\_t `PFlashBase`
- uint32\_t `PFlashSize`
- uint32\_t `DFlashBase`
- uint32\_t `EERAMBase`
- `flash_callback_t` `CallBack`
- uint32\_t `PFlashBase`
- uint32\_t `PFlashSize`
- uint32\_t `DFlashBase`
- uint32\_t `DFlashSize`
- uint32\_t `EERAMBase`
- uint32\_t `EEESize`
- `flash_callback_t` `CallBack`
- uint8\_t `brownOutCode`
- uint16\_t `numOfRecordReqMaintain`
- uint16\_t `sectorEraseCount`

### 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.*

### 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` ((flash\_callback\_t)0xFFFFFFFFU)  
*Null callback.*

### Flash driver APIs

- status\_t `FLASH_DRV_Init` (const flash\_user\_config\_t \*const pUserConf, flash\_ssd\_config\_t \*const pSSDConfig)  
*Initializes Flash.*
- void `FLASH_DRV_GetPFlashProtection` (uint32\_t \*protectStatus)  
*P-Flash get protection.*
- status\_t `FLASH_DRV_SetPFlashProtection` (uint32\_t protectStatus)  
*P-Flash set protection.*
- void `FLASH_DRV_GetSecurityState` (uint8\_t \*securityState)  
*Flash get security state.*
- status\_t `FLASH_DRV_SecurityBypass` (const flash\_ssd\_config\_t \*pSSDConfig, const uint8\_t \*keyBuffer)  
*Flash security bypass.*
- status\_t `FLASH_DRV_EraseAllBlock` (const flash\_ssd\_config\_t \*pSSDConfig)  
*Flash erase all blocks.*
- status\_t `FLASH_DRV_VerifyAllBlock` (const flash\_ssd\_config\_t \*pSSDConfig, uint8\_t marginLevel)  
*Flash verify all blocks.*
- status\_t `FLASH_DRV_EraseSector` (const flash\_ssd\_config\_t \*pSSDConfig, uint32\_t dest, uint32\_t size)  
*Flash erase sector.*
- status\_t `FLASH_DRV_VerifySection` (const flash\_ssd\_config\_t \*pSSDConfig, uint32\_t dest, uint16\_t number, uint8\_t marginLevel)  
*Flash verify section.*
- void `FLASH_DRV_EraseSuspend` (void)  
*Flash erase suspend.*

- void [FLASH\\_DRV\\_EraseResume](#) (void)  
*Flash erase resume.*
- status\_t [FLASH\\_DRV\\_ReadOnce](#) (const [flash\\_ssd\\_config\\_t](#) \*pSSDConfig, uint8\_t recordIndex, uint8\_t \*p←  
dataArray)  
*Flash read once.*
- status\_t [FLASH\\_DRV\\_ProgramOnce](#) (const [flash\\_ssd\\_config\\_t](#) \*pSSDConfig, uint8\_t recordIndex, const  
uint8\_t \*pDataArray)  
*Flash program once.*
- status\_t [FLASH\\_DRV\\_Program](#) (const [flash\\_ssd\\_config\\_t](#) \*pSSDConfig, uint32\_t dest, uint32\_t size, const  
uint8\_t \*pData)  
*Flash program.*
- status\_t [FLASH\\_DRV\\_ProgramCheck](#) (const [flash\\_ssd\\_config\\_t](#) \*pSSDConfig, uint32\_t dest, uint32\_t size,  
const uint8\_t \*pExpectedData, uint32\_t \*pFailAddr, uint8\_t marginLevel)  
*Flash program check.*
- status\_t [FLASH\\_DRV\\_CheckSum](#) (const [flash\\_ssd\\_config\\_t](#) \*pSSDConfig, uint32\_t dest, uint32\_t size,  
uint32\_t \*pSum)  
*Calculates check sum.*
- status\_t [FLASH\\_DRV\\_EnableCmdCompleteInterrupt](#) (void)  
*Enable the command complete interrupt.*
- void [FLASH\\_DRV\\_DisableCmdCompleteInterrupt](#) (void)  
*Disable the command complete interrupt.*
- static bool [FLASH\\_DRV\\_GetCmdCompleteFlag](#) (void)  
*Check the command complete flag has completed or not.*
- status\_t [FLASH\\_DRV\\_EnableReadCollisionInterrupt](#) (void)  
*Enable the read collision error interrupt.*
- void [FLASH\\_DRV\\_DisableReadCollisionInterrupt](#) (void)  
*Disable the read collision error interrupt.*
- static bool [FLASH\\_DRV\\_GetReadCollisionFlag](#) (void)  
*Check the read collision error flag is detected or not.*
- static void [FLASH\\_DRV\\_ClearReadCollisionFlag](#) (void)  
*Clear the read collision error flag.*
- void [FLASH\\_DRV\\_GetDefaultConfig](#) ([flash\\_user\\_config\\_t](#) \*const config)  
*Get default flash configuration.*

## 16.48.2 Data Structure Documentation

### 16.48.2.1 struct flash\_user\_config\_t

Flash User Configuration Structure.

Implements : [flash\\_user\\_config\\_t\\_Class](#)

Definition at line 786 of file [flash\\_driver.h](#).

#### Data Fields

- uint32\_t [PFlashBase](#)
- uint32\_t [PFlashSize](#)
- uint32\_t [DFlashBase](#)
- uint32\_t [EERAMBase](#)
- [flash\\_callback\\_t](#) [Callback](#)

### 16.48.2.2 struct flash\_ssd\_config\_t

Flash SSD Configuration Structure.

The structure includes the static parameters for C90TFS/FTFx which are device-dependent. The fields including PFlashBlockBase, PFlashBlockSize, DFlashBlockBase, EERAMBlockBase, and CallBack are passed via [flash\\_user\\_config\\_t](#). The rest of parameters such as DFlashBlockSize, and EEESize will be initialized in [FLASH\\_DRV\\_Init\(\)](#) automatically.

Implements : flash\_ssd\_config\_t\_Class

Definition at line 810 of file flash\_driver.h.

#### Data Fields

- uint32\_t [PFlashBase](#)
- uint32\_t [PFlashSize](#)
- uint32\_t [DFlashBase](#)
- uint32\_t [DFlashSize](#)
- uint32\_t [EERAMBase](#)
- uint32\_t [EEESize](#)
- [flash\\_callback\\_t](#) CallBack

### 16.48.2.3 struct flash\_eeprom\_status\_t

EEPROM status structure.

Implements : flash\_eeprom\_status\_t\_Class

Definition at line 832 of file flash\_driver.h.

#### Data Fields

- uint8\_t [brownOutCode](#)
- uint16\_t [numOfRecordReqMaintain](#)
- uint16\_t [sectorEraseCount](#)

## 16.48.3 Macro Definition Documentation

**16.48.3.1** #define CLEAR\_FTFx\_FSTAT\_ERROR\_BITS FTFx\_FSTAT = (uint8\_t)(FTFx\_FSTAT\_FPVIOL\_MASK | FTFx\_FSTAT\_ACCERR\_MASK | FTFx\_FSTAT\_RDCOLERR\_MASK)

Definition at line 602 of file flash\_driver.h.

**16.48.3.2** #define CSE\_KEY\_SIZE\_CODE\_MAX 0x03U

Definition at line 699 of file flash\_driver.h.

**16.48.3.3** #define DFLASH\_IFR\_READRESOURCE\_ADDRESS 0x8000FCU

Definition at line 681 of file flash\_driver.h.

**16.48.3.4** #define FLASH\_CALLBACK\_CS 0x0AU

Callback period count for FlashCheckSum.

This value is only relevant for FlashCheckSum operation, where a high rate of calling back can impair performance. The rest of the flash operations invoke the callback as often as possible while waiting for the flash controller to finish the requested operation.

Definition at line 744 of file flash\_driver.h.

**16.48.3.5 #define FLASH\_NOT\_SECURE 0x01U**

Flash currently not in secure state.

Definition at line 727 of file flash\_driver.h.

**16.48.3.6 #define FLASH\_SECURE\_BACKDOOR\_DISABLED 0x04U**

Flash is secured and backdoor key access disabled.

Definition at line 731 of file flash\_driver.h.

**16.48.3.7 #define FLASH\_SECURE\_BACKDOOR\_ENABLED 0x02U**

Flash is secured and backdoor key access enabled.

Definition at line 729 of file flash\_driver.h.

**16.48.3.8 #define FLASH\_SECURITY\_STATE\_KEYEN 0x80U**

Definition at line 690 of file flash\_driver.h.

**16.48.3.9 #define FLASH\_SECURITY\_STATE\_UNSECURED 0x02U**

Definition at line 691 of file flash\_driver.h.

**16.48.3.10 #define FTFx\_DPHRASE\_SIZE 0x0010U**

Definition at line 611 of file flash\_driver.h.

**16.48.3.11 #define FTFx\_ERASE\_ALL\_BLOCK 0x44U**

Definition at line 635 of file flash\_driver.h.

**16.48.3.12 #define FTFx\_ERASE\_ALL\_BLOCK\_UNSECURE 0x49U**

Definition at line 638 of file flash\_driver.h.

**16.48.3.13 #define FTFx\_ERASE\_BLOCK 0x08U**

Definition at line 629 of file flash\_driver.h.

**16.48.3.14 #define FTFx\_ERASE\_SECTOR 0x09U**

Definition at line 630 of file flash\_driver.h.

**16.48.3.15 #define FTFx\_FSTAT\_ERROR\_BITS (FTFx\_FSTAT & (FTFx\_FSTAT\_MGSTAT0\_MASK | FTFx\_FSTAT\_FPVIOL\_MASK | FTFx\_FSTAT\_ACCERR\_MASK | FTFx\_FSTAT\_RDCOLERR\_MASK))**

Definition at line 700 of file flash\_driver.h.

**16.48.3.16 #define FTFx\_LONGWORD\_SIZE 0x0004U**

Definition at line 607 of file flash\_driver.h.

**16.48.3.17 #define FTFx\_PFLASH\_SWAP 0x46U**

Definition at line 637 of file flash\_driver.h.

**16.48.3.18 #define FTFx\_PHRASE\_SIZE 0x0008U**

Definition at line 609 of file flash\_driver.h.

16.48.3.19 **#define FTFx\_PROGRAM\_CHECK 0x02U**

Definition at line 625 of file flash\_driver.h.

16.48.3.20 **#define FTFx\_PROGRAM\_LONGWORD 0x06U**

Definition at line 627 of file flash\_driver.h.

16.48.3.21 **#define FTFx\_PROGRAM\_ONCE 0x43U**

Definition at line 634 of file flash\_driver.h.

16.48.3.22 **#define FTFx\_PROGRAM\_PARTITION 0x80U**

Definition at line 639 of file flash\_driver.h.

16.48.3.23 **#define FTFx\_PROGRAM\_PHRASE 0x07U**

Definition at line 628 of file flash\_driver.h.

16.48.3.24 **#define FTFx\_PROGRAM\_SECTION 0x0BU**

Definition at line 631 of file flash\_driver.h.

16.48.3.25 **#define FTFx\_READ\_ONCE 0x41U**

Definition at line 633 of file flash\_driver.h.

16.48.3.26 **#define FTFx\_READ\_RESOURCE 0x03U**

Definition at line 626 of file flash\_driver.h.

16.48.3.27 **#define FTFx\_RSRC\_CODE\_REG FTFx\_FCCOB8**

Definition at line 617 of file flash\_driver.h.

16.48.3.28 **#define FTFx\_SECURITY\_BY\_PASS 0x45U**

Definition at line 636 of file flash\_driver.h.

16.48.3.29 **#define FTFx\_SET\_EERAM 0x81U**

Definition at line 640 of file flash\_driver.h.

16.48.3.30 **#define FTFx\_SWAP\_COMPLETE 0x04U**

Complete swap mode.

Definition at line 670 of file flash\_driver.h.

16.48.3.31 **#define FTFx\_SWAP\_READY 0x01U**

Ready swap mode.

Definition at line 664 of file flash\_driver.h.

16.48.3.32 **#define FTFx\_SWAP\_REPORT\_STATUS 0x08U**

Report Swap Status.

Definition at line 654 of file flash\_driver.h.

**16.48.3.33** `#define FTFx_SWAP_SET_IN_COMPLETE 0x04U`

Set Swap in Complete State.

Definition at line 652 of file flash\_driver.h.

**16.48.3.34** `#define FTFx_SWAP_SET_IN_PREPARE 0x02U`

Set Swap in Update State.

Definition at line 650 of file flash\_driver.h.

**16.48.3.35** `#define FTFx_SWAP_SET_INDICATOR_ADDR 0x01U`

Initialize Swap System control code.

Definition at line 648 of file flash\_driver.h.

**16.48.3.36** `#define FTFx_SWAP_UNINIT 0x00U`

Uninitialized swap mode.

Definition at line 662 of file flash\_driver.h.

**16.48.3.37** `#define FTFx_SWAP_UPDATE 0x02U`

Update swap mode.

Definition at line 666 of file flash\_driver.h.

**16.48.3.38** `#define FTFx_SWAP_UPDATE_ERASED 0x03U`

Update-Erased swap mode.

Definition at line 668 of file flash\_driver.h.

**16.48.3.39** `#define FTFx_VERIFY_ALL_BLOCK 0x40U`

Definition at line 632 of file flash\_driver.h.

**16.48.3.40** `#define FTFx_VERIFY_BLOCK 0x00U`

Definition at line 623 of file flash\_driver.h.

**16.48.3.41** `#define FTFx_VERIFY_SECTION 0x01U`

Definition at line 624 of file flash\_driver.h.

**16.48.3.42** `#define FTFx_WORD_SIZE 0x0002U`

Definition at line 605 of file flash\_driver.h.

**16.48.3.43** `#define GET_BIT_0_7( value ) ((uint8_t)((uint32_t)(value)) & 0xFFU)`

Definition at line 684 of file flash\_driver.h.

**16.48.3.44** `#define GET_BIT_16_23( value ) ((uint8_t)((((uint32_t)(value)) >> 16) & 0xFFU))`

Definition at line 686 of file flash\_driver.h.

**16.48.3.45** `#define GET_BIT_24_31( value ) ((uint8_t)((((uint32_t)(value)) >> 24))`

Definition at line 687 of file flash\_driver.h.

16.48.3.46 **#define GET\_BIT\_8\_15( value ) (((uint8\_t)((((uint32\_t)(value)) >> 8) & 0xFFU))**

Definition at line 685 of file flash\_driver.h.

16.48.3.47 **#define NULL\_CALLBACK ((flash\_callback\_t)0xFFFFFFFFU)**

Null callback.

Definition at line 755 of file flash\_driver.h.

16.48.3.48 **#define RESUME\_WAIT\_CNT 0x20U**

Resume wait count used in FLASH\_DRV\_EraseResume function.

Definition at line 674 of file flash\_driver.h.

16.48.3.49 **#define SUSPEND\_WAIT\_CNT 0x40U**

Suspend wait count used in FLASH\_DRV\_EraseSuspend function.

Definition at line 676 of file flash\_driver.h.

## 16.48.4 Typedef Documentation

16.48.4.1 **typedef void(\* flash\_callback\_t) (void)**

Call back function pointer data type.

If using callback in the application, any code reachable from this function must not be placed in a Flash block targeted for a program/erase operation. Functions can be placed in RAM section by using the START/END\_FUNCTION\_DEFINITION/DECLARATION\_RAMSECTION macros.

Definition at line 772 of file flash\_driver.h.

## 16.48.5 Enumeration Type Documentation

16.48.5.1 **enum flash\_flexRam\_function\_control\_code\_t**

FlexRAM Function control Code.

Implements : flash\_flexRAM\_function\_control\_code\_t\_Class

### Enumerator

**EEE\_ENABLE** Make FlexRAM available for emulated EEPROM

**EEE\_QUICK\_WRITE** Make FlexRAM available for EEPROM quick writes

**EEE\_STATUS\_QUERY** EEPROM quick write status query

**EEE\_COMPLETE\_INTERRUPT\_QUICK\_WRITE** Complete interrupted EEPROM quick write process

**EEE\_DISABLE** Make FlexRAM available as RAM

Definition at line 713 of file flash\_driver.h.

## 16.48.6 Function Documentation

16.48.6.1 **status\_t FLASH\_DRV\_CheckSum ( const flash\_ssd\_config\_t \* pSSDConfig, uint32\_t dest, uint32\_t size, uint32\_t \* pSum )**

Calculates check sum.



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 [flash\\_driver.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 STATUS\_SUCCESS if size provided by user is zero regardless of the input validation.

#### Parameters

in	<i>pSSDConfig</i>	The SSD configuration structure pointer.
in	<i>dest</i>	Start address of the Flash range to be summed.
in	<i>size</i>	Size in byte of the Flash range to be summed.
in	<i>pSum</i>	To return the sum value.

#### Returns

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failure was occurred.

Definition at line 1041 of file flash\_driver.c.

**16.48.6.2** `static void FLASH_DRV_ClearReadCollisionFlag ( void ) [inline],[static]`

Clear the read collision error flag.

Implements : FLASH\_DRV\_ClearReadCollisionFlag\_Activity

Definition at line 1690 of file flash\_driver.h.

**16.48.6.3** `void FLASH_DRV_DisableCmdCompleteInterrupt ( void )`

Disable the command complete interrupt.

Definition at line 2004 of file flash\_driver.c.

**16.48.6.4** `void FLASH_DRV_DisableReadCollisionInterrupt ( void )`

Disable the read collision error interrupt.

Definition at line 2039 of file flash\_driver.c.

**16.48.6.5** `status_t FLASH_DRV_EnableCmdCompleteInterrupt ( void )`

Enable the command complete interrupt.

This function will enable the command complete interrupt is generated when an FTFC command completes.

#### Returns

operation status

- STATUS\_SUCCESS: Operation was successful.

Definition at line 1986 of file flash\_driver.c.

**16.48.6.6** `status_t FLASH_DRV_EnableReadCollisionInterrupt ( void )`

Enable the read collision error interrupt.

This function will enable the read collision error interrupt generation when an FTFC read collision error occurs.

**Returns**

operation status

- STATUS\_SUCCESS: Operation was successful.

Definition at line 2021 of file flash\_driver.c.

**16.48.6.7** status\_t FLASH\_DRV\_EraseAllBlock ( const flash\_ssd\_config\_t \* pSSDConfig )

Flash erase all blocks.

This API erases all Flash memory, initializes the FlexRAM, verifies all memory contents, and then releases the MCU security.

**Parameters**

in	pSSDConfig	The SSD configuration structure pointer.
----	------------	--

**Returns**

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failure was occurred.
- STATUS\_BUSY: Operation was busy.

Definition at line 461 of file flash\_driver.c.

**16.48.6.8** void FLASH\_DRV\_EraseResume ( void )

Flash erase resume.

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.

Definition at line 707 of file flash\_driver.c.

**16.48.6.9** status\_t FLASH\_DRV\_EraseSector ( const flash\_ssd\_config\_t \* pSSDConfig, uint32\_t dest, uint32\_t size )

Flash erase sector.

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.

## Parameters

in	pSSDConfig	The SSD configuration structure pointer.								
in	dest	Address in the first sector to be erased. User need to make sure the dest address in of P-FLASH or D-FLASH block. This address should be aligned to bytes following a below table								
		FL↔ ASH TY↔ P↔ E/↔ M↔ CU	S32↔ K116	S32↔ K118	S32↔ K142	S32↔ K144	S32↔ K146	S32↔ K148	S32↔ K142↔ W	S32↔ K144↔ W
		P↔ FL↔ ASH	8	8	8	16	16	16	8	16
		D↔ FL↔ ASH	8	8	8	8	8	16	8	8
in	size	Size to be erased in bytes. It is used to determine number of sectors to be erased. This size should be aligned to bytes following a below table								
		FL↔ ASH TY↔ P↔ E/↔ M↔ CU	S32↔ K116	S32↔ K118	S32↔ K142	S32↔ K144	S32↔ K146	S32↔ K148	S32↔ K142↔ W	S32↔ K144↔ W
		P↔ FL↔ ASH	8	8	8	16	16	16	8	16
		D↔ FL↔ ASH	8	8	8	8	8	16	8	8

## Returns

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failure was occurred.
- STATUS\_BUSY: Operation was busy.

Definition at line 531 of file flash\_driver.c.

## 16.48.6.10 void FLASH\_DRV\_EraseSuspend ( void )

Flash erase suspend.

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.

Definition at line 682 of file flash\_driver.c.

## 16.48.6.11 static bool FLASH\_DRV\_GetCmdCompleteFlag ( void ) [inline],[static]

Check the command complete flag has completed or not.

**Returns**

the command complete flag

- true: The FTFC command has completed.
- false: The FTFC command is in progress.

Implements : FLASH\_DRV\_GetCmdCompleteFlag\_Activity

Definition at line 1649 of file flash\_driver.h.

#### 16.48.6.12 void FLASH\_DRV\_GetDefaultConfig ( flash\_user\_config\_t \*const config )

Get default flash configuration.

This API will get default flash user configuration.

**Parameters**

out	config	Pointer flash user configuration structure.
-----	--------	---

Definition at line 2191 of file flash\_driver.c.

#### 16.48.6.13 void FLASH\_DRV\_GetPFlashProtection ( uint32\_t \* protectStatus )

P-Flash get protection.

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**

out	protectStatus	<p>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:</p> <ul style="list-style-type: none"> <li>• 0: this area is protected.</li> <li>• 1: this area is unprotected.</li> </ul>
-----	---------------	---

Definition at line 314 of file flash\_driver.c.

#### 16.48.6.14 static bool FLASH\_DRV\_GetReadCollisionFlag ( void ) [inline],[static]

Check the read collision error flag is detected or not.

**Returns**

the read collision error flag

- true: Collision error detected.
- false: No collision error detected.

Implements : FLASH\_DRV\_GetReadCollisionFlag\_Activity

Definition at line 1680 of file flash\_driver.h.

#### 16.48.6.15 void FLASH\_DRV\_GetSecurityState ( uint8\_t \* securityState )

Flash get security state.

This API retrieves the current Flash security status, including the security enabling state and the back door key enabling state.

## Parameters

out	<i>securityState</i>	To return the current security status code. <ul style="list-style-type: none"> <li>FLASH_NOT_SECURE (0x01U): Flash currently not in secure state</li> <li>FLASH_SECURE_BACKDOOR_ENABLED (0x02U): Flash is secured and back door key access enabled</li> <li>FLASH_SECURE_BACKDOOR_DISABLED (0x04U): Flash is secured and back door key access disabled.</li> </ul>
-----	----------------------	--

Definition at line 378 of file flash\_driver.c.

**16.48.6.16** `status_t FLASH_DRV_Init ( const flash_user_config_t *const pUserConf, flash_ssd_config_t *const pSSDConfig )`

Initializes Flash.

This API initializes Flash module by reporting the memory configuration via SSD configuration structure.

## Parameters

in	<i>pUserConf</i>	The user configuration structure pointer.
in	<i>pSSDConfig</i>	The SSD configuration structure pointer.

## Returns

operation status

- STATUS\_SUCCESS: Operation was successful.

Definition at line 225 of file flash\_driver.c.

**16.48.6.17** `status_t FLASH_DRV_Program ( const flash_ssd_config_t * pSSDConfig, uint32_t dest, uint32_t size, const uint8_t * pData )`

Flash program.

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

in	<i>pSSDConfig</i>	The SSD configuration structure pointer.
in	<i>dest</i>	Start address for the intended program operation. This address should be aligned to 8 bytes.
in	<i>size</i>	Size in byte to be programmed. This size should be aligned to 8 bytes.
in	<i>pData</i>	Pointer of source address from which data has to be taken for program operation.

## Returns

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failure was occurred.
- STATUS\_BUSY: Operation was busy.

Definition at line 908 of file flash\_driver.c.

**16.48.6.18** `status_t FLASH_DRV_ProgramCheck ( const flash_ssd_config_t * pSSDConfig, uint32_t dest, uint32_t size, const uint8_t * pExpectedData, uint32_t * pFailAddr, uint8_t marginLevel )`

Flash program check.

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

in	<i>pSSDConfig</i>	The SSD configuration structure pointer.
in	<i>dest</i>	Start address for the intended program check operation. This address should be aligned to 4 bytes.
in	<i>size</i>	Size in byte to check accuracy of program operation. This size should be aligned to 4 bytes.
in	<i>pExpectedData</i>	The pointer to the expected data.
in	<i>pFailAddr</i>	Returned the first aligned failing address.
in	<i>marginLevel</i>	Read margin choice as follows: <ul style="list-style-type: none"> <li>marginLevel = 0x1U: read at User margin 1/0 level.</li> <li>marginLevel = 0x2U: read at Factory margin 1/0 level.</li> </ul>

#### Returns

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failure was occurred.
- STATUS\_BUSY: Operation was busy.

Definition at line 1000 of file flash\_driver.c.

**16.48.6.19** `status_t FLASH_DRV_ProgramOnce ( const flash_ssd_config_t * pSSDConfig, uint8_t recordIndex, const uint8_t * pDataArray )`

Flash program once.

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

in	<i>pSSDConfig</i>	The SSD configuration structure pointer.
in	<i>recordIndex</i>	The record index will be read. It can be from 0x0U to 0x7U or from 0x0U to 0xF according to specific derivative.
in	<i>pdataArray</i>	Pointer to the array from which data will be taken for program once command.

#### Returns

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failure was occurred.
- STATUS\_BUSY: Operation was busy.

Definition at line 783 of file flash\_driver.c.

**16.48.6.20** `status_t FLASH_DRV_ReadOnce ( const flash_ssd_config_t * pSSDConfig, uint8_t recordIndex, uint8_t * pDataArray )`

Flash read once.

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

in	<i>pSSDConfig</i>	The SSD configuration structure pointer.
in	<i>recordIndex</i>	The record index will be read. It can be from 0x0U to 0x7U or from 0x0U to 0xF according to specific derivative.
in	<i>pdataArray</i>	Pointer to the array to return the data read by the read once command.

#### Returns

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failure was occurred.
- STATUS\_BUSY: Operation was busy.

Definition at line 732 of file flash\_driver.c.

**16.48.6.21** `status_t FLASH_DRV_SecurityBypass ( const flash_ssd_config_t * pSSDConfig, const uint8_t * keyBuffer )`

Flash security bypass.

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

in	<i>pSSDConfig</i>	The SSD configuration structure pointer.
in	<i>keyBuffer</i>	Point to the user buffer containing the back door key.

#### Returns

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failure was occurred.
- STATUS\_BUSY: Operation was busy.

Definition at line 420 of file flash\_driver.c.

**16.48.6.22** `status_t FLASH_DRV_SetPFlashProtection ( uint32_t protectStatus )`

P-Flash set protection.

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 will not be changed.

#### Parameters

in	<i>protectStatus</i>	<p>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:</p> <ul style="list-style-type: none"> <li>• 0: this area is protected.</li> <li>• 1: this area is unprotected.</li> </ul>
----	----------------------	---

**Returns**

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failure was occurred.

Definition at line 337 of file flash\_driver.c.

**16.48.6.23** `status_t FLASH_DRV_VerifyAllBlock ( const flash_ssd_config_t * pSSDConfig, uint8_t marginLevel )`

Flash verify all blocks.

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**

in	<i>pSSDConfig</i>	The SSD configuration structure pointer.
in	<i>marginLevel</i>	Read Margin Choice as follows: <ul style="list-style-type: none"> <li>• marginLevel = 0x0U: use the Normal read level</li> <li>• marginLevel = 0x1U: use the User read</li> <li>• marginLevel = 0x2U: use the Factory read</li> </ul>

**Returns**

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failure was occurred.
- STATUS\_BUSY: Operation was busy.

Definition at line 495 of file flash\_driver.c.

**16.48.6.24** `status_t FLASH_DRV_VerifySection ( const flash_ssd_config_t * pSSDConfig, uint32_t dest, uint16_t number, uint8_t marginLevel )`

Flash verify section.

This API checks if a section of the P-Flash or the D-Flash memory is erased to the specified read margin level.



## Parameters

in	pSSDConfig	The SSD configuration structure pointer.								
in	dest	Start address for the intended verify operation. User need to make sure the dest address in of P-FLASH or D-FLASH block. This address should be aligned to bytes following a below table								
		FL↔ ASH↔ TY↔ P↔ E/↔ M↔ CU	S32↔ K116	S32↔ K118	S32↔ K142	S32↔ K144	S32↔ K146	S32↔ K148	S32↔ K142↔ W	S32↔ K144↔ W
		P-↔ FL↔ ASH	8	8	8	16	16	16	8	16
		D-↔ FL↔ ASH	8	8	8	8	8	16	8	8
in	number	Number of alignment unit to be verified. Refer to corresponding reference manual to get correct information of alignment constrain.								
in	marginLevel	Read Margin Choice as follows: <ul style="list-style-type: none"><li>• marginLevel = 0x0U: use Normal read level</li><li>• marginLevel = 0x1U: use the User read</li><li>• marginLevel = 0x2U: use the Factory read</li></ul>								

## Returns

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failure was occurred.
- STATUS\_BUSY: Operation was busy.

Definition at line 612 of file flash\_driver.c.

## 16.48.7 Variable Documentation

## 16.48.7.1 uint8\_t brownOutCode

Brown-out detection code

Definition at line 834 of file flash\_driver.h.

## 16.48.7.2 flash\_callback\_t Callback

Call back function to service the time critical events. Any code reachable from this function must not be placed in a Flash block targeted for a program/erase operation

Definition at line 794 of file flash\_driver.h.

## 16.48.7.3 flash\_callback\_t Callback

Call back function to service the time critical events. Any code reachable from this function must not be placed in a Flash block targeted for a program/erase operation

Definition at line 823 of file flash\_driver.h.

**16.48.7.4 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

Definition at line 790 of file flash\_driver.h.

**16.48.7.5 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

Definition at line 814 of file flash\_driver.h.

**16.48.7.6 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

Definition at line 816 of file flash\_driver.h.

**16.48.7.7 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

Definition at line 821 of file flash\_driver.h.

**16.48.7.8 uint32\_t EERAMBase**

The base address of FlexRAM (for FlexNVM device) or acceleration RAM memory (for non-FlexNVM device)

Definition at line 792 of file flash\_driver.h.

**16.48.7.9 uint32\_t EERAMBase**

The base address of FlexRAM (for FlexNVM device) or acceleration RAM memory (for non-FlexNVM device)

Definition at line 819 of file flash\_driver.h.

**16.48.7.10 uint16\_t numOfRecordReqMaintain**

Number of EEPROM quick write records requiring maintenance

Definition at line 835 of file flash\_driver.h.

**16.48.7.11 uint32\_t PFlashBase**

The base address of P-Flash memory

Definition at line 788 of file flash\_driver.h.

**16.48.7.12 uint32\_t PFlashBase**

The base address of P-Flash memory

Definition at line 812 of file flash\_driver.h.

**16.48.7.13 uint32\_t PFlashSize**

The size in byte of P-Flash memory

Definition at line 789 of file flash\_driver.h.

**16.48.7.14 uint32\_t PFlashSize**

The size in byte of P-Flash memory

Definition at line 813 of file flash\_driver.h.

**16.48.7.15 uint16\_t sectorEraseCount**

EEPROM sector erase count

Definition at line 836 of file flash\_driver.h.

## 16.49 Flash Memory (Flash)

### 16.49.1 Detailed Description

Flash Memory Module provides the general flash APIs.

Flash memory is ideal for single-supply applications, permitting in-the-field erase and reprogramming operations without the need for any external high voltage power sources. The flash module includes a memory controller that executes commands to modify flash memory contents. An erased bit reads '1' and a programmed bit reads '0'. The programming operation is unidirectional; it can only move bits from the '1' state (erased) to the '0' state (programmed). Only the erase operation restores bits from '0' to '1'; bits cannot be programmed from a '0' to a '1'.

#### C90TFS Flash Driver

The C90TFS flash module includes the following accessible memory regions.

1. Program flash memory for vector space and code store.
2. FlexNVM for data store, additional code store and also non-volatile storage for the EEPROM filing system representing data written to the FlexRAM requiring highest endurance.
3. FlexRAM for high-endurance EEPROM data store or traditional RAM.

Some platforms may be designed to have only program flash memory or all of them.

The S32 SDK provides the C90TFS Flash driver of S32K platforms. The driver includes general APIs to handle specific operations on C90TFS Flash module. The user can use those APIs directly in the application.

#### EEPROM feature

For platforms with FlexNVM, the flash module provides a built-in hardware emulation scheme to emulate the characteristics of an EEPROM by effectively providing a high-endurance, byte write-able NVM. The EEPROM system is shown in the following figure.

*Figure 1. EEPROM Architecture*

To handle with various customer's requirements, the FlexRAM and FlexNVM blocks can be split into partitions:

1. EEPROM partition(EESIZE) — The amount of FlexRAM used for EEPROM can be set from 0 Bytes (no EEPROM) to the maximum FlexRAM size. The remainder of the FlexRAM not used for EEPROM is not accessible while the FlexRAM is configured for EEPROM. The EEPROM partition grows upward from the bottom of the FlexRAM address space.
2. Data flash partition(DEPART) — The amount of FlexNVM memory used for data flash can be programmed from 0 bytes (all of the FlexNVM block is available for EEPROM backup) to the maximum size of the FlexNVM block.
3. FlexNVM EEPROM partition — The amount of FlexNVM memory used for EEPROM backup, which is equal to the FlexNVM block size minus the data flash memory partition size. The EEPROM backup size must be at least 16 times the EEPROM partition size in FlexRAM.

The partition information (EESIZE, DEPART) is programmed using the **#FLASH\_DRV\_DEFlashPartition** API.

The function of FlexRAM can be changed from EEPROM usage to traditional RAM for accelerate programming in **#FLASH\_DRV\_ProgramSection** API and vice versa by **#FLASH\_DRV\_SetFlexRamFunction** API.

This is example code of EEE usage sequence:

```
/* Provide information about the flash blocks. */
const flash_user_config_t Flash_InitConfig0 = {
    .PFlashBase = 0x00000000U, /* Base address of Program Flash block */
    .PFlashSize = 0x00100000U, /* Size of Program Flash block */
    .DFlashBase = 0x10000000U, /* Base address of Data Flash block */
}
```

```

.EERAMBase = 0x14000000U, /* Base address of FlexRAM block */
/* If using callback, any code reachable from this function must not be placed in a Flash block
   targeted for a program/erase operation.*/
.CallBack = NULL_CALLBACK
};

/* Declare a FLASH configuration structure which is initialized by FlashInit, and will be used by all
   flash APIs */
flash_ssd_config_t flashSSDConfig;
status_t ret; /* Store the driver APIs return code */

/* Data source for program operation */
#define BUFFER_SIZE 0x100u /* Size of data source */
#define SIZE_WRITE_EEE 0x7u /* Size of data write in EEPROM */
uint8_t sourceBuffer[BUFFER_SIZE];

/* Init source data */
for (i = 0u; i < BUFFER_SIZE; i++)
{
    sourceBuffer[i] = i;
}

/* Always initialize the driver before calling other functions */
ret = FLASH_DRV_Init(&Flash_InitConfig0, &flashSSDConfig);
if (ret != STATUS_SUCCESS)
{
    return ret;
}

#if ((FEATURE_FLS_HAS_FLEX_NVM == 1u) & (FEATURE_FLS_HAS_FLEX_RAM == 1u))
/* Configure FlexRAM as EEPROM if it is currently used as traditional RAM */
if (flashSSDConfig.EEESize == 0u)
{
    /* Configure FlexRAM as EEPROM and FlexNVM as EEPROM backup region,
       DEFflashPartition will be failed if the IFR region isn't blank.
       Refer to the device document for valid EEPROM Data Size Code
       and FlexNVM Partition Code. For example on S32K144:
       - EEEDataSizeCode = 0x02u: EEPROM size = 4 Kbytes
       - DEPartitionCode = 0x08u: EEPROM backup size = 64 Kbytes */
    ret = FLASH_DRV_DEFflashPartition(&flashSSDConfig, 0x02u, 0x08u, 0x0, false, true);
    DEV_ASSERT(STATUS_SUCCESS == ret);
    else
    {
        /* Re-initialize the driver to update the new EEPROM configuration */
        ret = FLASH_DRV_Init(&Flash_InitConfig0, &flashSSDConfig);
        if (ret != STATUS_SUCCESS)
        {
            return ret;
        }

        /* Make FlexRAM available for EEPROM */
        ret = FLASH_DRV_SetFlexRamFunction(&flashSSDConfig, EEE_ENABLE, 0x0u, NULL);
        DEV_ASSERT(STATUS_SUCCESS == ret);
    }
}
else /* FLeXRAM is already configured as EEPROM */
{
    /* Make FlexRAM available for EEPROM, make sure that FlexNVM and FlexRAM
       are already partitioned successfully before */
    ret = FLASH_DRV_SetFlexRamFunction(&flashSSDConfig, EEE_ENABLE, 0x0u, NULL);
    DEV_ASSERT(STATUS_SUCCESS == ret);
}
#endif

/* Erase the sixth PFlash sector */
/* Configure address, size to erase sector function. For example on S32K144 */
address = 6u * FEATURE_FLS_PF_BLOCK_SECTOR_SIZE; /* A sector size is 4KB */
size = FEATURE_FLS_PF_BLOCK_SECTOR_SIZE;
ret = FLASH_DRV_EraseSector(&flashSSDConfig, address, size);
DEV_ASSERT(STATUS_SUCCESS == ret);

/* Verify the erase operation at margin level value of 1, user read */
ret = FLASH_DRV_VerifySection(&flashSSDConfig, address, size, 1u);
DEV_ASSERT(STATUS_SUCCESS == ret);

/* Write some data to the erased PFlash sector */
size = BUFFER_SIZE;
ret = FLASH_DRV_Program(&flashSSDConfig, address, size, sourceBuffer);
DEV_ASSERT(STATUS_SUCCESS == ret);

/* Verify the program operation at margin level value of 1, user margin */
ret = FLASH_DRV_ProgramCheck(&flashSSDConfig, address, size, sourceBuffer, &
    failAddr, 1u);
DEV_ASSERT(STATUS_SUCCESS == ret);

/* Try to write data to EEPROM if FlexRAM is configured as EEPROM */
if (flashSSDConfig.EEESize != 0u)

```

```

{
    address = flashSSDConfig.EERAMBase;
    size = SIZE_WRITE_EEE;
    ret = FLASH_DRV_EEEwrite(&flashSSDConfig, address, size, sourceBuffer);
    DEV_ASSERT(STATUS_SUCCESS == ret);
}

```

### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\drivers\src\flash\flash_driver.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc\
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

#### Clock Manager

#### Important Note

1. If using callback in the application, any code reachable from this function must not be placed in a Flash block targeted for a program/erase operation to avoid the RWW error. Functions can be placed in RAM section by using the START/END\_FUNCTION\_DEFINITION/DECLARATION\_RAMSECTION macros.
2. To suspend the sector erase operation for a simple method, invoke the **FLASH\_DRV\_EraseSuspend** function within callback of **FLASH\_DRV\_EraseSector**. In this case, the **FLASH\_DRV\_EraseSuspend** must not be placed in the same block in which the Flash erase sector command is going on.
3. **#FLASH\_DRV\_CommandSequence**, **FLASH\_DRV\_EraseSuspend** and **FLASH\_DRV\_EraseResume** should be executed from RAM or different Flash blocks which are targeted for writing to avoid the RWW error. **FLASH\_DRV\_EraseSuspend** and **FLASH\_DRV\_EraseResume** functions should be called in pairs.
4. To guarantee the correct execution of this driver, the Flash cache in the Flash memory controller module should be disabled before invoking any API.
5. Partitioning FlexNVM and FlexRAM for EEPROM usage shall be executed only once in the lifetime of the device.
6. After successfully partitioning FlexNVM and FlexRAM for EEPROM usage, user needs to call **FLASH\_DRV\_V\_Init** to update memory information in global structure.
7. Can not erase or program flash when MCU is high speed run mode or very low power mode.
8. S32K14xW needs to enable FlexRAM as traditional RAM for the first time using silicon.

#### Modules

- **Flash Memory (Flash)**

## 16.50 FlexCAN Driver

### 16.50.1 Detailed Description

#### How to use the FlexCAN driver in your application

In order to be able to use the FlexCAN in your application, the first thing to do is initializing it with the desired configuration. This is done by calling the **FLEXCAN\_DRV\_Init** function. One of the arguments passed to this function is the configuration which will be used for the FlexCAN module, specified by the [flexcan\\_user\\_config\\_t](#) structure.

The [flexcan\\_user\\_config\\_t](#) structure allows you to configure the following:

- the number of message buffers needed;
- the number of Rx FIFO ID filters needed;
- enable/disable the Rx FIFO feature;
- the operation mode, which can be one of the following:
  - normal mode;
  - listen-only mode;
  - loopback mode;
  - freeze mode;
  - disable mode;
- the payload size of the message buffers:
  - 8 bytes;
  - 16 bytes (only available with the FD feature enabled);
  - 32 bytes (only available with the FD feature enabled);
  - 64 bytes (only available with the FD feature enabled);
- enable/disable the Flexible Data-rate feature;
- the clock source of the CAN Protocol Engine (PE);
- the bitrate used for standard frames or for the arbitration phase of FD frames;
- the bitrate used for the data phase of FD frames;
- the Rx FIFO transfer type, which can be one of the following:
  - using interrupts;
  - using DMA, only on supported platforms;
- the DMA channel number to be used for DMA transfers, only on supported platforms;

The bitrate is represented by a [flexcan\\_time\\_segment\\_t](#) structure, with the following fields:

- propagation segment;
- phase segment 1;
- phase segment 2;
- clock prescaler division factor;
- resync jump width.

Details about these fields can be found in the reference manual.

In order to use a mailbox for transmission/reception, it should be initialized using either **FLEXCAN\_DRV\_Config**, **RxMb**, **FLEXCAN\_DRV\_ConfigRxFifo** or **FLEXCAN\_DRV\_ConfigTxMb**.

After having the mailbox configured, you can start sending/receiving data using the specified mailbox, by calling one of the following functions:

- FLEXCAN\_DRV\_Send;
- FLEXCAN\_DRV\_SendBlocking;
- FLEXCAN\_DRV\_Receive;
- FLEXCAN\_DRV\_ReceiveBlocking;
- FLEXCAN\_DRV\_RxFifo;
- FLEXCAN\_DRV\_RxFifoBlocking.

A default FlexCAN configuration can be accessed by calling the **FLEXCAN\_DRV\_GetDefaultConfig** function. This function takes as argument a **flexcan\_user\_config\_t** structure and fills it according to the following settings:

- 16 message buffers
- flexible data rate disabled
- Rx FIFO disabled
- normal operation mode
- 8 byte payload size
- Protocol Engine clock = Oscillator clock
- bitrate of 500 Kbit/s (computed for PE clock = 8 MHz with sample point = 87.5)

#### FlexCAN Rx FIFO configuration

The Rx FIFO is receive-only and 6-message deep. The user can read the received messages sequentially, in the order they were received, by repeatedly reading Message Buffer 0 (zero). The Rx FIFO ID filter table (configurable from 8 to 128 table elements) specifies filtering criteria for accepting frames into the FIFO. This table is represented through a structure of **flexcan\_id\_table\_t** type, which specifies if Remote Frames are accepted into the FIFO if they match the target ID, whether extended or standard frames are accepted into the FIFO if they match the target ID and the target ID.

```
/* ID Filter table */
const flexcan_id_table_t filterTable[] = {
{
    .isExtendedFrame = false,
    .isRemoteFrame = false,
    .id = 1U
},
...
};

FLEXCAN_DRV_ConfigRxFifo(INST_CANCOM1,
    FLEXCAN_RX_FIFO_ID_FORMAT_A, filterTable);
```

The number of elements in the ID filter table is defined by the following formula:

- for format A: the number of Rx FIFO ID filters
- for format B: twice the number of Rx FIFO ID filters
- for format C: four times the number of Rx FIFO ID filters The user must provide the exact number of elements in order to avoid any misconfiguration.



Each element in the ID filter table specifies an ID to be used as acceptance criteria for the FIFO, as follows:

- for format A: In the standard frame format, bits 10 to 0 of the ID are used for frame identification. In the extended frame format, bits 28 to 0 are used.
- for format B: In the standard frame format, bits 10 to 0 of the ID are used for frame identification. In the extended frame format, only the 14 most significant bits (28 to 15) of the ID are compared to the 14 most significant bits (28 to 15) of the received ID.
- for format C: In both standard and extended frame formats, only the 8 most significant bits (7 to 0 for standard, 28 to 21 for extended) of the ID are compared to the 8 most significant bits (7 to 0 for standard, 28 to 21 for extended) of the received ID.

When Rx FIFO feature is enabled, buffer 0 (zero) cannot be used for transmission. The transfer status in case of FIFO enable feature can be monitored by calling **FLEXCAN\_DRV\_GetTransferStatus** for buffer index 0.

In order to use Rx FIFO filter mask options, enabled by **FLEXCAN\_DRV\_SetRxIndividualMask()** and **FLEXCAN\_DRV\_SetRxFifoGlobalMask()** user needs to call **FLEXCAN\_DRV\_ConfigRxFifo()** before using these functions in Rx FIFO mode. In case of Rx FIFO ID filter format B or C the **FLEXCAN\_DRV\_SetRxFifoGlobalMask()** will apply the same mask for all filters IDs.

The **FLEXCAN\_DRV\_SetRxIndividualMask()** can self determine if CAN is in normal mode and will only set acceptance ID Mask. If CAN is in Rx FIFO mode, will determine the ID format type and will set the acceptance ID Mask as corresponding Id Filter Format corresponding to individual mask number the user must ensure that the ID Element is not affected by Rx FIFO Global Mask in this case the ID Filter will be set as normal configuration to allow functionality of receiving as normal MB of the remaining MBs outside of Rx FIFO use.

#### Important Notes

- The FlexCAN driver does not handle clock setup or any kind of pin configuration. This is handled by the **Clock Manager** and **PinSettings** modules, respectively. The driver assumes that the correct clock configurations have been made, so it is the user's responsibility to set up clocking and pin configurations correctly.
- For some platforms, the clock source of the CAN Protocol Engine (PE) is not configurable from the FlexCAN module. If this feature is not supported, the *pe\_clock* field from the FlexCAN configuration structure is not present.
- DMA module has to be initialized prior to FlexCAN Rx FIFO usage in DMA mode; also, the DMA channel needs to be allocated by the application (the driver only takes care of configuring the DMA channel received in the configuration structure).
- When used **FLEXCAN\_DRV\_ReceiveBlocking()** and **FLEXCAN\_DRV\_SendBlocking()** with timeout parameter 0 and the message is already in mailbox configured will report timeout and successful transmit or receive the message.
- For Cortex-M0 architecture S32K116 and S32K118 CPUs need to pass as transmission/reception buffers memory aligned, the only allowed exceptions are for **FLEXCAN\_DRV\_Send()**, **FLEXCAN\_DRV\_SendBlocking()**, **FLEXCAN\_DRV\_ConfigRemoteResponseMb** with a payload length less than 3 bytes
- When used the Pretended Network Mode, in Stop Mode the Interface Clock(CHI) (from Clock\_Manager) need to be disabled and Protocol Engine (PE) clock source enabled as selected from Can Module. Be aware that on wakeup Run Mode to have enabled the Interface Clock(CHI).
- In case inside the callback function is called another blocking reception (**FLEXCAN\_DRV\_ReceiveBlocking/FLEXCAN\_DRV\_RxFifoBlocking**) or abort **FLEXCAN\_DRV\_AbortTransfer** without polling previous operation status this can lead to undetermined behavior.

#### Example: ####

```
#define INST_CANCOM1 (0U)
#define RX_MAILBOX (1U)
#define MSG_ID (2U)

flexcan_state_t canCom1_State;
```

```

const flexcan_user_config_t canCom1_InitConfig0 = {
    .fd_enable = true,
    .pe_clock = FLEXCAN_CLK_SOURCE_OSC,
    .max_num_mb = 16,
    .num_id_filters = FLEXCAN_RX_FIFO_ID_FILTERS_8,
    .is_rx_fifo_needed = false,
    .flexcanMode = FLEXCAN_NORMAL_MODE,
    .payload = FLEXCAN_PAYLOAD_SIZE_8,
    .bitrate = {
        .propSeg = 7,
        .phaseSeg1 = 4,
        .phaseSeg2 = 1,
        .preDivider = 0,
        .rJumpwidth = 1
    },
    .bitrate_cbt = {
        .propSeg = 11,
        .phaseSeg1 = 1,
        .phaseSeg2 = 1,
        .preDivider = 0,
        .rJumpwidth = 1
    },
    .transfer_type = FLEXCAN_RX_FIFO_USING_INTERRUPTS,
    .rxFifoDMAChannel = 0U
};

/* Initialize FlexCAN driver */
FLEXCAN_DRV_Init(INST_CANCOM1, &canCom1_State, &canCom1_InitConfig0);

/* Set information about the data to be received */
flexcan_data_info_t dataInfo =
{
    .data_length = 1U,
    .msg_id_type = FLEXCAN_MSG_ID_STD,
    .enable_brs = true,
    .fd_enable = true,
    .fd_padding = 0U
};

/* Configure Rx message buffer with index 1 to receive frames with ID 2 */
FLEXCAN_DRV_ConfigRxMb(INST_CANCOM1, RX_MAILBOX, &dataInfo, MSG_ID);

/* Receive a frame in the recvBuff variable */
flexcan_msgbuff_t recvBuff;

FLEXCAN_DRV_Receive(INST_CANCOM1, RX_MAILBOX, &recvBuff);
/* Wait for the message to be received */
while (FLEXCAN_DRV_GetTransferStatus(INST_CANCOM1, RX_MAILBOX) == STATUS_BUSY)
    ;

/* De-initialize driver */
FLEXCAN_DRV_Deinit(INST_CANCOM1);

```

## Integration guideline

### Compilation units

The following files need to be compiled in the project:

```

${S32SDK_PATH}\platform\drivers\src\flexcan\flexcan_driver.c
${S32SDK_PATH}\platform\drivers\src\flexcan\flexcan_hw_access.c
${S32SDK_PATH}\platform\drivers\src\flexcan\flexcan_irq.c

```

### Include path

The following paths need to be added to the include path of the toolchain:

```

${S32SDK_PATH}\platform\drivers\inc\

```

### Compile symbols

No special symbols are required for this component

### Dependencies

[Clock Manager Interrupt Manager \(Interrupt\) Enhanced Direct Memory Access \(eDMA\)](#)

## Data Structures

- struct `flexcan_msgbuff_t`  
*FlexCAN message buffer structure Implements : `flexcan_msgbuff_t_Class`. [More...](#)*
- struct `flexcan_mb_handle_t`  
*Information needed for internal handling of a given MB. Implements : `flexcan_mb_handle_t_Class`. [More...](#)*
- struct `FlexCANState`  
*Internal driver state information. [More...](#)*
- struct `flexcan_data_info_t`  
*FlexCAN data info from user Implements : `flexcan_data_info_t_Class`. [More...](#)*
- struct `flexcan_id_table_t`  
*FlexCAN Rx FIFO ID filter table structure Implements : `flexcan_id_table_t_Class`. [More...](#)*
- struct `flexcan_time_segment_t`  
*FlexCAN bitrate related structures Implements : `flexcan_time_segment_t_Class`. [More...](#)*
- struct `flexcan_user_config_t`  
*FlexCAN configuration. [More...](#)*

## Typedefs

- typedef struct `FlexCANState flexcan_state_t`  
*Internal driver state information.*
- typedef void(\* `flexcan_callback_t`) (uint8\_t instance, `flexcan_event_type_t` eventType, uint32\_t buffIdx, `flexcan_state_t` \*flexcanState)  
*FlexCAN Driver callback function type Implements : `flexcan_callback_t_Class`.*
- typedef void(\* `flexcan_error_callback_t`) (uint8\_t instance, `flexcan_event_type_t` eventType, `flexcan_state_t` \*flexcanState)  
*FlexCAN Driver error callback function type Implements : `flexcan_error_callback_t_Class`.*

## Enumerations

- enum `flexcan_rxfifo_transfer_type_t` { `FLEXCAN_RXFIFO_USING_INTERRUPTS` }  
*The type of the RxFIFO transfer (interrupts/DMA). Implements : `flexcan_rxfifo_transfer_type_t_Class`.*
- enum `flexcan_event_type_t` { `FLEXCAN_EVENT_RX_COMPLETE`, `FLEXCAN_EVENT_RXFIFO_COMPLETE`, `FLEXCAN_EVENT_RXFIFO_WARNING`, `FLEXCAN_EVENT_RXFIFO_OVERFLOW`, `FLEXCAN_EVENT_TX_COMPLETE`, `FLEXCAN_EVENT_ERROR` }  
*The type of the event which occurred when the callback was invoked. Implements : `flexcan_event_type_t_Class`.*
- enum `flexcan_mb_state_t` { `FLEXCAN_MB_IDLE`, `FLEXCAN_MB_RX_BUSY`, `FLEXCAN_MB_TX_BUSY` }  
*The state of a given MB (idle/Rx busy/Tx busy). Implements : `flexcan_mb_state_t_Class`.*
- enum `flexcan_msgbuff_id_type_t` { `FLEXCAN_MSG_ID_STD`, `FLEXCAN_MSG_ID_EXT` }  
*FlexCAN Message Buffer ID type Implements : `flexcan_msgbuff_id_type_t_Class`.*
- enum `flexcan_rx_fifo_id_filter_num_t` { `FLEXCAN_RX_FIFO_ID_FILTERS_8` = 0x0, `FLEXCAN_RX_FIFO_ID_FILTERS_16` = 0x1, `FLEXCAN_RX_FIFO_ID_FILTERS_24` = 0x2, `FLEXCAN_RX_FIFO_ID_FILTERS_32` = 0x3, `FLEXCAN_RX_FIFO_ID_FILTERS_40` = 0x4, `FLEXCAN_RX_FIFO_ID_FILTERS_48` = 0x5, `FLEXCAN_RX_FIFO_ID_FILTERS_56` = 0x6, `FLEXCAN_RX_FIFO_ID_FILTERS_64` = 0x7, `FLEXCAN_RX_FIFO_ID_FILTERS_72` = 0x8, `FLEXCAN_RX_FIFO_ID_FILTERS_80` = 0x9, `FLEXCAN_RX_FIFO_ID_FILTERS_88` = 0xA, `FLEXCAN_RX_FIFO_ID_FILTERS_96` = 0xB, `FLEXCAN_RX_FIFO_ID_FILTERS_104` = 0xC, `FLEXCAN_RX_FIFO_ID_FILTERS_112` = 0xD, `FLEXCAN_RX_FIFO_ID_FILTERS_120` = 0xE, `FLEXCAN_RX_FIFO_ID_FILTERS_128` = 0xF }  
*FlexCAN Rx FIFO filters number Implements : `flexcan_rx_fifo_id_filter_num_t_Class`.*
- enum `flexcan_rx_mask_type_t` { `FLEXCAN_RX_MASK_GLOBAL`, `FLEXCAN_RX_MASK_INDIVIDUAL` }

*FlexCAN Rx mask type. Implements : flexcan\_rx\_mask\_type\_t Class.*

- enum [flexcan\\_rx\\_fifo\\_id\\_element\\_format\\_t](#) { [FLEXCAN\\_RX\\_FIFO\\_ID\\_FORMAT\\_A](#), [FLEXCAN\\_RX\\_FIFO\\_ID\\_FORMAT\\_B](#), [FLEXCAN\\_RX\\_FIFO\\_ID\\_FORMAT\\_C](#), [FLEXCAN\\_RX\\_FIFO\\_ID\\_FORMAT\\_D](#) }

*ID formats for Rx FIFO Implements : flexcan\_rx\_fifo\_id\_element\_format\_t Class.*

- enum [flexcan\\_operation\\_modes\\_t](#) { [FLEXCAN\\_NORMAL\\_MODE](#), [FLEXCAN\\_LISTEN\\_ONLY\\_MODE](#), [FLEXCAN\\_LOOPBACK\\_MODE](#), [FLEXCAN\\_FREEZE\\_MODE](#), [FLEXCAN\\_DISABLE\\_MODE](#) }

*FlexCAN operation modes Implements : flexcan\_operation\_modes\_t Class.*

#### Bit rate

- void [FLEXCAN\\_DRV\\_SetBitrate](#) (uint8\_t instance, const [flexcan\\_time\\_segment\\_t](#) \*bitrate)  
*Sets the FlexCAN bit rate for standard frames or the arbitration phase of FD frames.*
- void [FLEXCAN\\_DRV\\_GetBitrate](#) (uint8\_t instance, [flexcan\\_time\\_segment\\_t](#) \*bitrate)  
*Gets the FlexCAN bit rate for standard frames or the arbitration phase of FD frames.*

#### Rx MB and Rx FIFO masks

- void [FLEXCAN\\_DRV\\_SetRxMaskType](#) (uint8\_t instance, [flexcan\\_rx\\_mask\\_type\\_t](#) type)  
*Sets the Rx masking type.*
- void [FLEXCAN\\_DRV\\_SetRxFifoGlobalMask](#) (uint8\_t instance, [flexcan\\_msgbuff\\_id\\_type\\_t](#) id\_type, uint32\_t mask)  
*Sets the FlexCAN Rx FIFO global mask (standard or extended). This mask is applied to all filters ID regardless the ID Filter format.*
- void [FLEXCAN\\_DRV\\_SetRxMbGlobalMask](#) (uint8\_t instance, [flexcan\\_msgbuff\\_id\\_type\\_t](#) id\_type, uint32\_t mask)  
*Sets the FlexCAN Rx MB global mask (standard or extended).*
- void [FLEXCAN\\_DRV\\_SetRxMb14Mask](#) (uint8\_t instance, [flexcan\\_msgbuff\\_id\\_type\\_t](#) id\_type, uint32\_t mask)  
*Sets the FlexCAN Rx MB 14 mask (standard or extended).*
- void [FLEXCAN\\_DRV\\_SetRxMb15Mask](#) (uint8\_t instance, [flexcan\\_msgbuff\\_id\\_type\\_t](#) id\_type, uint32\_t mask)  
*Sets the FlexCAN Rx MB 15 mask (standard or extended).*
- status\_t [FLEXCAN\\_DRV\\_SetRxIndividualMask](#) (uint8\_t instance, [flexcan\\_msgbuff\\_id\\_type\\_t](#) id\_type, uint8\_t mb\_idx, uint32\_t mask)  
*Sets the FlexCAN Rx individual mask (standard or extended).*

#### Initialization and Shutdown

- uint32\_t [FLEXCAN\\_DRV\\_GetDefaultConfig](#) ([flexcan\\_user\\_config\\_t](#) \*config)  
*Gets the default configuration structure.*
- status\_t [FLEXCAN\\_DRV\\_Init](#) (uint8\_t instance, [flexcan\\_state\\_t](#) \*state, const [flexcan\\_user\\_config\\_t](#) \*data)  
*Initializes the FlexCAN peripheral.*
- status\_t [FLEXCAN\\_DRV\\_Deinit](#) (uint8\_t instance)  
*Shuts down a FlexCAN instance.*

#### Send configuration

- status\_t [FLEXCAN\\_DRV\\_ConfigTxMb](#) (uint8\_t instance, uint8\_t mb\_idx, const [flexcan\\_data\\_info\\_t](#) \*tx\_info, uint32\_t msg\_id)  
*FlexCAN transmit message buffer field configuration.*
- status\_t [FLEXCAN\\_DRV\\_ConfigRemoteResponseMb](#) (uint8\_t instance, uint8\_t mb\_idx, const [flexcan\\_data\\_info\\_t](#) \*tx\_info, uint32\_t msg\_id, const uint8\_t \*mb\_data)

*Configures a transmit message buffer for remote frame response.*

- status\_t [FLEXCAN\\_DRV\\_SendBlocking](#) (uint8\_t instance, uint8\_t mb\_idx, const [flexcan\\_data\\_info\\_t](#) \*tx\_info, uint32\_t msg\_id, const uint8\_t \*mb\_data, uint32\_t timeout\_ms)

*Sends a CAN frame using the specified message buffer, in a blocking manner.*

- status\_t [FLEXCAN\\_DRV\\_Send](#) (uint8\_t instance, uint8\_t mb\_idx, const [flexcan\\_data\\_info\\_t](#) \*tx\_info, uint32\_t msg\_id, const uint8\_t \*mb\_data)

*Sends a CAN frame using the specified message buffer.*

## Receive configuration

- status\_t [FLEXCAN\\_DRV\\_ConfigRxMb](#) (uint8\_t instance, uint8\_t mb\_idx, const [flexcan\\_data\\_info\\_t](#) \*rx\_info, uint32\_t msg\_id)

*FlexCAN receive message buffer field configuration.*

- void [FLEXCAN\\_DRV\\_ConfigRxFifo](#) (uint8\_t instance, [flexcan\\_rx\\_fifo\\_id\\_element\\_format\\_t](#) id\_format, const [flexcan\\_id\\_table\\_t](#) \*id\_filter\_table)

*FlexCAN Rx FIFO field configuration.*

- status\_t [FLEXCAN\\_DRV\\_ReceiveBlocking](#) (uint8\_t instance, uint8\_t mb\_idx, [flexcan\\_msgbuff\\_t](#) \*data, uint32\_t timeout\_ms)

*Receives a CAN frame using the specified message buffer, in a blocking manner.*

- status\_t [FLEXCAN\\_DRV\\_Receive](#) (uint8\_t instance, uint8\_t mb\_idx, [flexcan\\_msgbuff\\_t](#) \*data)

*Receives a CAN frame using the specified message buffer.*

- status\_t [FLEXCAN\\_DRV\\_RxFifoBlocking](#) (uint8\_t instance, [flexcan\\_msgbuff\\_t](#) \*data, uint32\_t timeout\_ms)

*Receives a CAN frame using the message FIFO, in a blocking manner.*

- status\_t [FLEXCAN\\_DRV\\_RxFifo](#) (uint8\_t instance, [flexcan\\_msgbuff\\_t](#) \*data)

*Receives a CAN frame using the message FIFO.*

## Transfer status

- status\_t [FLEXCAN\\_DRV\\_AbortTransfer](#) (uint8\_t instance, uint8\_t mb\_idx)

*Ends a non-blocking FlexCAN transfer early.*

- status\_t [FLEXCAN\\_DRV\\_GetTransferStatus](#) (uint8\_t instance, uint8\_t mb\_idx)

*Returns whether the previous FlexCAN transfer has finished.*

- uint32\_t [FLEXCAN\\_DRV\\_GetErrorStatus](#) (uint8\_t instance)

*Returns reported error conditions.*

## IRQ handler callback

- void [FLEXCAN\\_DRV\\_InstallEventCallback](#) (uint8\_t instance, [flexcan\\_callback\\_t](#) callback, void \*callbackParam)

*Installs a callback function for the IRQ handler.*

- void [FLEXCAN\\_DRV\\_InstallErrorCallback](#) (uint8\_t instance, [flexcan\\_error\\_callback\\_t](#) callback, void \*callbackParam)

*Installs an error callback function for the IRQ handler and enables error interrupts.*

## 16.50.2 Data Structure Documentation

### 16.50.2.1 struct flexcan\_msgbuff\_t

FlexCAN message buffer structure Implements : flexcan\_msgbuff\_t\_Class.

Definition at line 100 of file flexcan\_driver.h.

**Data Fields**

- [uint32\\_t cs](#)
- [uint32\\_t msgId](#)
- [uint8\\_t data](#) [64]
- [uint8\\_t dataLen](#)

**Field Documentation****16.50.2.1.1 uint32\_t cs****Code and Status**

Definition at line 101 of file flexcan\_driver.h.

**16.50.2.1.2 uint8\_t data[64]**

Data bytes of the FlexCAN message

Definition at line 103 of file flexcan\_driver.h.

**16.50.2.1.3 uint8\_t dataLen**

Length of data in bytes

Definition at line 104 of file flexcan\_driver.h.

**16.50.2.1.4 uint32\_t msgId**

Message Buffer ID

Definition at line 102 of file flexcan\_driver.h.

**16.50.2.2 struct flexcan\_mb\_handle\_t**

Information needed for internal handling of a given MB. Implements : flexcan\_mb\_handle\_t\_Class.

Definition at line 110 of file flexcan\_driver.h.

**Data Fields**

- [flexcan\\_msgbuff\\_t \\* mb\\_message](#)
- [semaphore\\_t mbSema](#)
- [volatile flexcan\\_mb\\_state\\_t state](#)
- [bool isBlocking](#)
- [bool isRemote](#)

**Field Documentation****16.50.2.2.1 bool isBlocking**

True if the transfer is blocking

Definition at line 114 of file flexcan\_driver.h.

**16.50.2.2.2 bool isRemote**

True if the frame is a remote frame

Definition at line 115 of file flexcan\_driver.h.

**16.50.2.2.3 flexcan\_msgbuff\_t\* mb\_message**

The FlexCAN MB structure

Definition at line 111 of file flexcan\_driver.h.

## 16.50.2.2.4 semaphore\_t mbSema

Semaphore used for signaling completion of a blocking transfer

Definition at line 112 of file flexcan\_driver.h.

## 16.50.2.2.5 volatile flexcan\_mb\_state\_t state

The state of the current MB (idle/Rx busy/Tx busy)

Definition at line 113 of file flexcan\_driver.h.

## 16.50.2.3 struct FlexCANState

Internal driver state information.

## 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. Implements : flexcan\_state\_t\_Class

Definition at line 126 of file flexcan\_driver.h.

## Data Fields

- flexcan\_mb\_handle\_t mbs [FEATURE\_CAN\_MAX\_MB\_NUM]
- void(\* callback)(uint8\_t instance, flexcan\_event\_type\_t eventType, uint32\_t buffIdx, struct FlexCANState \*driverState)
- void \* callbackParam
- void(\* error\_callback)(uint8\_t instance, flexcan\_event\_type\_t eventType, struct FlexCANState \*driverState)
- void \* errorCallbackParam
- flexcan\_rxfifo\_transfer\_type\_t transferType

## Field Documentation

## 16.50.2.3.1 void(\* callback)(uint8\_t instance, flexcan\_event\_type\_t eventType, uint32\_t buffIdx, struct FlexCANState \*driverState)

IRQ handler callback function.

Definition at line 129 of file flexcan\_driver.h.

## 16.50.2.3.2 void\* callbackParam

Parameter used to pass user data when invoking the callback function.

Definition at line 133 of file flexcan\_driver.h.

## 16.50.2.3.3 void(\* error\_callback)(uint8\_t instance, flexcan\_event\_type\_t eventType, struct FlexCANState \*driverState)

Error IRQ handler callback function.

Definition at line 136 of file flexcan\_driver.h.

## 16.50.2.3.4 void\* errorCallbackParam

Parameter used to pass user data when invoking the error callback function.

Definition at line 140 of file flexcan\_driver.h.

## 16.50.2.3.5 flexcan\_mb\_handle\_t mbs[FEATURE\_CAN\_MAX\_MB\_NUM]

Array containing information related to each MB

Definition at line 127 of file flexcan\_driver.h.

#### 16.50.2.3.6 flexcan\_rxfifo\_transfer\_type\_t transferType

Type of RxFIFO transfer.

Definition at line 147 of file flexcan\_driver.h.

#### 16.50.2.4 struct flexcan\_data\_info\_t

FlexCAN data info from user Implements : flexcan\_data\_info\_t\_Class.

Definition at line 153 of file flexcan\_driver.h.

##### Data Fields

- [flexcan\\_msgbuff\\_id\\_type\\_t msg\\_id\\_type](#)
- [uint32\\_t data\\_length](#)
- [bool is\\_remote](#)

##### Field Documentation

##### 16.50.2.4.1 uint32\_t data\_length

Length of Data in Bytes

Definition at line 155 of file flexcan\_driver.h.

##### 16.50.2.4.2 bool is\_remote

Specifies if the frame is standard or remote

Definition at line 162 of file flexcan\_driver.h.

##### 16.50.2.4.3 flexcan\_msgbuff\_id\_type\_t msg\_id\_type

Type of message ID (standard or extended)

Definition at line 154 of file flexcan\_driver.h.

#### 16.50.2.5 struct flexcan\_id\_table\_t

FlexCAN Rx FIFO ID filter table structure Implements : flexcan\_id\_table\_t\_Class.

Definition at line 209 of file flexcan\_driver.h.

##### Data Fields

- [bool isRemoteFrame](#)
- [bool isExtendedFrame](#)
- [uint32\\_t id](#)

##### Field Documentation

##### 16.50.2.5.1 uint32\_t id

Rx FIFO ID filter element

Definition at line 212 of file flexcan\_driver.h.

##### 16.50.2.5.2 bool isExtendedFrame

Extended frame

Definition at line 211 of file flexcan\_driver.h.



#### 16.50.2.5.3 bool isRemoteFrame

Remote frame

Definition at line 210 of file flexcan\_driver.h.

#### 16.50.2.6 struct flexcan\_time\_segment\_t

FlexCAN bitrate related structures Implements : flexcan\_time\_segment\_t\_Class.

Definition at line 241 of file flexcan\_driver.h.

##### Data Fields

- uint32\_t [propSeg](#)
- uint32\_t [phaseSeg1](#)
- uint32\_t [phaseSeg2](#)
- uint32\_t [preDivider](#)
- uint32\_t [rJumpwidth](#)

##### Field Documentation

#### 16.50.2.6.1 uint32\_t phaseSeg1

Phase segment 1

Definition at line 243 of file flexcan\_driver.h.

#### 16.50.2.6.2 uint32\_t phaseSeg2

Phase segment 2

Definition at line 244 of file flexcan\_driver.h.

#### 16.50.2.6.3 uint32\_t preDivider

Clock prescaler division factor

Definition at line 245 of file flexcan\_driver.h.

#### 16.50.2.6.4 uint32\_t propSeg

Propagation segment

Definition at line 242 of file flexcan\_driver.h.

#### 16.50.2.6.5 uint32\_t rJumpwidth

Resync jump width

Definition at line 246 of file flexcan\_driver.h.

#### 16.50.2.7 struct flexcan\_user\_config\_t

FlexCAN configuration.

Definition at line 253 of file flexcan\_driver.h.

##### Data Fields

- uint32\_t [max\\_num\\_mb](#)
- [flexcan\\_rx\\_fifo\\_id\\_filter\\_num\\_t](#) num\_id\_filters
- bool [is\\_rx\\_fifo\\_needed](#)
- [flexcan\\_operation\\_modes\\_t](#) flexcanMode
- [flexcan\\_time\\_segment\\_t](#) bitrate
- [flexcan\\_rxfifo\\_transfer\\_type\\_t](#) transfer\_type

## Field Documentation

### 16.50.2.7.1 flexcan\_time\_segment\_t bitrate

The bitrate used for standard frames or for the arbitration phase of FD frames.

Definition at line 269 of file flexcan\_driver.h.

### 16.50.2.7.2 flexcan\_operation\_modes\_t flexcanMode

User configurable FlexCAN operation modes.

Definition at line 260 of file flexcan\_driver.h.

### 16.50.2.7.3 bool is\_rx\_fifo\_needed

1 if needed; 0 if not. This controls whether the Rx FIFO feature is enabled or not.

Definition at line 258 of file flexcan\_driver.h.

### 16.50.2.7.4 uint32\_t max\_num\_mb

The maximum number of Message Buffers

Definition at line 254 of file flexcan\_driver.h.

### 16.50.2.7.5 flexcan\_rx\_fifo\_id\_filter\_num\_t num\_id\_filters

The number of RX FIFO ID filters needed

Definition at line 256 of file flexcan\_driver.h.

### 16.50.2.7.6 flexcan\_rxfifo\_transfer\_type\_t transfer\_type

Specifies if the Rx FIFO uses interrupts or DMA.

Definition at line 273 of file flexcan\_driver.h.

## 16.50.3 Typedef Documentation

### 16.50.3.1 typedef void(\* flexcan\_callback\_t) (uint8\_t instance, flexcan\_event\_type\_t eventType, uint32\_t buffIdx, flexcan\_state\_t \*flexcanState)

FlexCAN Driver callback function type Implements : flexcan\_callback\_t\_Class.

Definition at line 335 of file flexcan\_driver.h.

### 16.50.3.2 typedef void(\* flexcan\_error\_callback\_t) (uint8\_t instance, flexcan\_event\_type\_t eventType, flexcan\_state\_t \*flexcanState)

FlexCAN Driver error callback function type Implements : flexcan\_error\_callback\_t\_Class.

Definition at line 341 of file flexcan\_driver.h.

### 16.50.3.3 typedef struct FlexCANState flexcan\_state\_t

Internal driver state information.

#### 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. Implements : flexcan\_state\_t\_Class

## 16.50.4 Enumeration Type Documentation

## 16.50.4.1 enum flexcan\_event\_type\_t

The type of the event which occurred when the callback was invoked. Implements : flexcan\_event\_type\_t\_Class.

## Enumerator

**FLEXCAN\_EVENT\_RX\_COMPLETE** A frame was received in the configured Rx MB.

**FLEXCAN\_EVENT\_RXFIFO\_COMPLETE** A frame was received in the Rx FIFO.

**FLEXCAN\_EVENT\_RXFIFO\_WARNING** Rx FIFO is almost full (5 frames).

**FLEXCAN\_EVENT\_RXFIFO\_OVERFLOW** Rx FIFO is full (incoming message was lost).

**FLEXCAN\_EVENT\_TX\_COMPLETE** A frame was sent from the configured Tx MB.

**FLEXCAN\_EVENT\_ERROR**

Definition at line 49 of file flexcan\_driver.h.

## 16.50.4.2 enum flexcan\_mb\_state\_t

The state of a given MB (idle/Rx busy/Tx busy). Implements : flexcan\_mb\_state\_t\_Class.

## Enumerator

**FLEXCAN\_MB\_IDLE** The MB is not used by any transfer.

**FLEXCAN\_MB\_RX\_BUSY** The MB is used for a reception.

**FLEXCAN\_MB\_TX\_BUSY** The MB is used for a transmission.

Definition at line 70 of file flexcan\_driver.h.

## 16.50.4.3 enum flexcan\_msgbuff\_id\_type\_t

FlexCAN Message Buffer ID type Implements : flexcan\_msgbuff\_id\_type\_t\_Class.

## Enumerator

**FLEXCAN\_MSG\_ID\_STD** Standard ID

**FLEXCAN\_MSG\_ID\_EXT** Extended ID

Definition at line 82 of file flexcan\_driver.h.

## 16.50.4.4 enum flexcan\_operation\_modes\_t

FlexCAN operation modes Implements : flexcan\_operation\_modes\_t\_Class.

## Enumerator

**FLEXCAN\_NORMAL\_MODE** Normal mode or user mode

**FLEXCAN\_LISTEN\_ONLY\_MODE** Listen-only mode

**FLEXCAN\_LOOPBACK\_MODE** Loop-back mode

**FLEXCAN\_FREEZE\_MODE** Freeze mode

**FLEXCAN\_DISABLE\_MODE** Module disable mode

Definition at line 218 of file flexcan\_driver.h.

## 16.50.4.5 enum flexcan\_rx\_fifo\_id\_element\_format\_t

ID formats for Rx FIFO Implements : flexcan\_rx\_fifo\_id\_element\_format\_t\_Class.

## Enumerator

- FLEXCAN\_RX\_FIFO\_ID\_FORMAT\_A** One full ID (standard and extended) per ID Filter Table element.  
**FLEXCAN\_RX\_FIFO\_ID\_FORMAT\_B** Two full standard IDs or two partial 14-bit (standard and extended) IDs per ID Filter Table element.  
**FLEXCAN\_RX\_FIFO\_ID\_FORMAT\_C** Four partial 8-bit Standard IDs per ID Filter Table element.  
**FLEXCAN\_RX\_FIFO\_ID\_FORMAT\_D** All frames rejected.

Definition at line 198 of file flexcan\_driver.h.

## 16.50.4.6 enum flexcan\_rx\_fifo\_id\_filter\_num\_t

FlexCAN Rx FIFO filters number Implements : flexcan\_rx\_fifo\_id\_filter\_num\_t\_Class.

## Enumerator

- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_8** 8 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_16** 16 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_24** 24 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_32** 32 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_40** 40 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_48** 48 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_56** 56 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_64** 64 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_72** 72 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_80** 80 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_88** 88 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_96** 96 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_104** 104 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_112** 112 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_120** 120 Rx FIFO Filters.  
**FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_128** 128 Rx FIFO Filters.

Definition at line 168 of file flexcan\_driver.h.

## 16.50.4.7 enum flexcan\_rx\_mask\_type\_t

FlexCAN Rx mask type. Implements : flexcan\_rx\_mask\_type\_t\_Class.

## Enumerator

- FLEXCAN\_RX\_MASK\_GLOBAL** Rx global mask  
**FLEXCAN\_RX\_MASK\_INDIVIDUAL** Rx individual mask

Definition at line 190 of file flexcan\_driver.h.

## 16.50.4.8 enum flexcan\_rxfifo\_transfer\_type\_t

The type of the RxFIFO transfer (interrupts/DMA). Implements : flexcan\_rxfifo\_transfer\_type\_t\_Class.

## Enumerator

- FLEXCAN\_RXFIFO\_USING\_INTERRUPTS** Use interrupts for RxFIFO.

Definition at line 39 of file flexcan\_driver.h.

## 16.50.5 Function Documentation

16.50.5.1 `status_t FLEXCAN_DRV_AbortTransfer ( uint8_t instance, uint8_t mb_idx )`

Ends a non-blocking FlexCAN transfer early.

## Parameters

<i>instance</i>	A FlexCAN instance number
<i>mb_idx</i>	The index of the message buffer

## Returns

STATUS\_SUCCESS if successful; STATUS\_CAN\_NO\_TRANSFER\_IN\_PROGRESS if no transfer was running

Definition at line 1950 of file flexcan\_driver.c.

16.50.5.2 `status_t FLEXCAN_DRV_ConfigRemoteResponseMb ( uint8_t instance, uint8_t mb_idx, const flexcan_data_info_t * tx_info, uint32_t msg_id, const uint8_t * mb_data )`

Configures a transmit message buffer for remote frame response.

## 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

## Returns

STATUS\_SUCCESS if successful; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the index of the message buffer is invalid

Definition at line 931 of file flexcan\_driver.c.

16.50.5.3 `void FLEXCAN_DRV_ConfigRxFifo ( uint8_t instance, flexcan_rx_fifo_id_element_format_t id_format, const flexcan_id_table_t * id_filter_table )`

FlexCAN Rx FIFO field configuration.

## Note

The number of elements in the ID filter table is defined by the following formula:

- for format A: the number of Rx FIFO ID filters
- for format B: twice the number of Rx FIFO ID filters
- for format C: four times the number of Rx FIFO ID filters The user must provide the exact number of elements in order to avoid any misconfiguration.

Each element in the ID filter table specifies an ID to be used as acceptance criteria for the FIFO as follows:

- for format A: In the standard frame format, bits 10 to 0 of the ID are used for frame identification. In the extended frame format, bits 28 to 0 are used.
- for format B: In the standard frame format, bits 10 to 0 of the ID are used for frame identification. In the extended frame format, only the 14 most significant bits (28 to 15) of the ID are compared to the 14 most significant bits (28 to 15) of the received ID.
- for format C: In both standard and extended frame formats, only the 8 most significant bits (7 to 0 for standard, 28 to 21 for extended) of the ID are compared to the 8 most significant bits (7 to 0 for standard, 28 to 21 for extended) of the received ID.

## 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

Definition at line 1175 of file flexcan\_driver.c.

**16.50.5.4** `status_t FLEXCAN_DRV_ConfigRxMb ( uint8_t instance, uint8_t mb_idx, const flexcan_data_info_t * rx_info, uint32_t msg_id )`

FlexCAN receive message buffer field configuration.

## 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

STATUS\_SUCCESS if successful; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the index of a message buffer is invalid;

Definition at line 1113 of file flexcan\_driver.c.

**16.50.5.5** `status_t FLEXCAN_DRV_ConfigTxMb ( uint8_t instance, uint8_t mb_idx, const flexcan_data_info_t * tx_info, uint32_t msg_id )`

FlexCAN transmit message buffer field configuration.

## 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

STATUS\_SUCCESS if successful; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the index of the message buffer is invalid

Definition at line 887 of file flexcan\_driver.c.

**16.50.5.6** `status_t FLEXCAN_DRV_Deinit ( uint8_t instance )`

Shuts down a FlexCAN instance.

## Parameters

<i>instance</i>	A FlexCAN instance number
-----------------	---------------------------

## Returns

STATUS\_SUCCESS if successful; STATUS\_ERROR if failed

Definition at line 1379 of file flexcan\_driver.c.

**16.50.5.7** `void FLEXCAN_DRV_GetBitrate ( uint8_t instance, flexcan_time_segment_t * bitrate )`

Gets the FlexCAN bit rate for standard frames or the arbitration phase of FD frames.

## Parameters

<i>instance</i>	A FlexCAN instance number
<i>bitrate</i>	A pointer to a variable for returning the FlexCAN bit rate settings

Definition at line 236 of file flexcan\_driver.c.

#### 16.50.5.8 uint32\_t FLEXCAN\_DRV\_GetDefaultConfig ( flexcan\_user\_config\_t \* config )

Gets the default configuration structure.

This function gets the default configuration structure, with the following settings:

- 16 message buffers
- flexible data rate disabled
- Rx FIFO disabled
- normal operation mode
- 8 byte payload size
- Protocol Engine clock = Oscillator clock
- bitrate of 500 Kbit/s (computed for sample point = 87.5)

## Parameters

out	<i>config</i>	The configuration structure
-----	---------------	-----------------------------

## Returns

The bitrate for generated configuration structure.

Definition at line 2759 of file flexcan\_driver.c.

#### 16.50.5.9 uint32\_t FLEXCAN\_DRV\_GetErrorStatus ( uint8\_t instance )

Returns reported error conditions.

Reports various error conditions detected in the reception and transmission of a CAN frame and some general status of the device.

## Parameters

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

## Returns

value of the Error and Status 1 register;

Definition at line 1931 of file flexcan\_driver.c.

#### 16.50.5.10 status\_t FLEXCAN\_DRV\_GetTransferStatus ( uint8\_t instance, uint8\_t mb\_idx )

Returns whether the previous FlexCAN transfer has finished.

When performing an async transfer, call this function to ascertain the state of the current transfer: in progress (or busy) or complete (success).

**Parameters**

<i>instance</i>	The FlexCAN instance number.
<i>mb_idx</i>	The index of the message buffer.

**Returns**

STATUS\_SUCCESS if successful; STATUS\_BUSY if a resource is busy; STATUS\_ERROR in case of a DMA error transfer;

Definition at line 1898 of file flexcan\_driver.c.

**16.50.5.11** `status_t FLEXCAN_DRV_Init ( uint8_t instance, flexcan_state_t * state, const flexcan_user_config_t * data )`

Initializes the FlexCAN peripheral.

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**

STATUS\_SUCCESS if successful; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the index of a message buffer is invalid; STATUS\_ERROR if other error occurred

Definition at line 687 of file flexcan\_driver.c.

**16.50.5.12** `void FLEXCAN_DRV_InstallErrorCallback ( uint8_t instance, flexcan_error_callback_t callback, void * callbackParam )`

Installs an error callback function for the IRQ handler and enables error interrupts.

**Parameters**

<i>instance</i>	The FlexCAN instance number.
<i>callback</i>	The error callback function.
<i>callbackParam</i>	User parameter passed to the error callback function through the state parameter.

Definition at line 2469 of file flexcan\_driver.c.

**16.50.5.13** `void FLEXCAN_DRV_InstallEventCallback ( uint8_t instance, flexcan_callback_t callback, void * callbackParam )`

Installs a callback function for the IRQ handler.

**Parameters**

<i>instance</i>	The FlexCAN instance number.
<i>callback</i>	The callback function.
<i>callbackParam</i>	User parameter passed to the callback function through the state parameter.

Definition at line 2449 of file flexcan\_driver.c.

**16.50.5.14** `status_t FLEXCAN_DRV_Receive ( uint8_t instance, uint8_t mb_idx, flexcan_msgbuff_t * data )`

Receives a CAN frame using the specified message buffer.

This function receives a CAN frame using a configured message buffer. The function returns immediately. If a callback is installed, it will be invoked after the frame was received and read into the specified buffer.



## 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

STATUS\_SUCCESS if successful; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the index of a message buffer is invalid; STATUS\_BUSY if a resource is busy

Definition at line 1268 of file flexcan\_driver.c.

**16.50.5.15** `status_t FLEXCAN_DRV_ReceiveBlocking ( uint8_t instance, uint8_t mb_idx, flexcan_msgbuff_t * data, uint32_t timeout_ms )`

Receives a CAN frame using the specified message buffer, in a blocking manner.

This function receives a CAN frame using a configured message buffer. The function blocks until either a frame was received, or the specified timeout expired.

## 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

STATUS\_SUCCESS if successful; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the index of a message buffer is invalid; STATUS\_BUSY if a resource is busy; STATUS\_TIMEOUT if the timeout is reached

Definition at line 1209 of file flexcan\_driver.c.

**16.50.5.16** `status_t FLEXCAN_DRV_RxFifo ( uint8_t instance, flexcan_msgbuff_t * data )`

Receives a CAN frame using the message FIFO.

This function receives a CAN frame using the Rx FIFO. The function returns immediately. If a callback is installed, it will be invoked after the frame was received and read into the specified buffer.

## Parameters

<i>instance</i>	A FlexCAN instance number
<i>data</i>	The FlexCAN receive message buffer data.

## Returns

STATUS\_SUCCESS if successful; STATUS\_BUSY if a resource is busy; STATUS\_ERROR if other error occurred

Definition at line 1358 of file flexcan\_driver.c.

**16.50.5.17** `status_t FLEXCAN_DRV_RxFifoBlocking ( uint8_t instance, flexcan_msgbuff_t * data, uint32_t timeout_ms )`

Receives a CAN frame using the message FIFO, in a blocking manner.

This function receives a CAN frame using the Rx FIFO. The function blocks until either a frame was received, or the specified timeout expired.

**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**

STATUS\_SUCCESS if successful; STATUS\_BUSY if a resource is busy; STATUS\_TIMEOUT if the timeout is reached; STATUS\_ERROR if other error occurred

Definition at line 1298 of file flexcan\_driver.c.

**16.50.5.18** `status_t FLEXCAN_DRV_Send ( uint8_t instance, uint8_t mb_idx, const flexcan_data_info_t * tx_info, uint32_t msg_id, const uint8_t * mb_data )`

Sends a CAN frame using the specified message buffer.

This function sends a CAN frame using a configured message buffer. The function returns immediately. If a callback is installed, it will be invoked after the frame was sent.

**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.

**Returns**

STATUS\_SUCCESS if successful; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the index of a message buffer is invalid; STATUS\_BUSY if a resource is busy

Definition at line 1070 of file flexcan\_driver.c.

**16.50.5.19** `status_t FLEXCAN_DRV_SendBlocking ( uint8_t instance, uint8_t mb_idx, const flexcan_data_info_t * tx_info, uint32_t msg_id, const uint8_t * mb_data, uint32_t timeout_ms )`

Sends a CAN frame using the specified message buffer, in a blocking manner.

This function sends a CAN frame using a configured message buffer. The function blocks until either the frame was sent, or the specified timeout expired.

**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**

STATUS\_SUCCESS if successful; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the index of a message buffer is invalid; STATUS\_BUSY if a resource is busy; STATUS\_TIMEOUT if the timeout is reached

Definition at line 968 of file flexcan\_driver.c.

**16.50.5.20** `void FLEXCAN_DRV_SetBitrate ( uint8_t instance, const flexcan_time_segment_t * bitrate )`

Sets the FlexCAN bit rate for standard frames or the arbitration phase of FD frames.

## Parameters

<i>instance</i>	A FlexCAN instance number
<i>bitrate</i>	A pointer to the FlexCAN bit rate settings.

Definition at line 159 of file flexcan\_driver.c.

**16.50.5.21** void FLEXCAN\_DRV\_SetRxFifoGlobalMask ( uint8\_t *instance*, flexcan\_msgbuff\_id\_type\_t *id\_type*, uint32\_t *mask* )

Sets the FlexCAN Rx FIFO global mask (standard or extended). This mask is applied to all filters ID regardless the ID Filter format.

## Parameters

<i>instance</i>	A FlexCAN instance number
<i>id_type</i>	Standard ID or extended ID mask type
<i>mask</i>	Mask Value. In FIFO mode, when using ID Format A or B, bit 31 encodes RTR check and bit 30 encodes IDE check respectively. For ID Format C, bits 31 and 30 are ignored.

Definition at line 317 of file flexcan\_driver.c.

**16.50.5.22** status\_t FLEXCAN\_DRV\_SetRxIndividualMask ( uint8\_t *instance*, flexcan\_msgbuff\_id\_type\_t *id\_type*, uint8\_t *mb\_idx*, uint32\_t *mask* )

Sets the FlexCAN Rx individual mask (standard or extended).

## 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. In FIFO mode, when using ID Format A or B, bit 31 encodes RTR check and bit 30 encodes IDE check respectively. For ID Format C, bits 31 and 30 are ignored.

## Returns

STATUS\_SUCCESS if successful; STATUS\_CAN\_BUFF\_OUT\_OF\_RANGE if the index of the message buffer is invalid.

Definition at line 504 of file flexcan\_driver.c.

**16.50.5.23** void FLEXCAN\_DRV\_SetRxMaskType ( uint8\_t *instance*, flexcan\_rx\_mask\_type\_t *type* )

Sets the Rx masking type.

## Parameters

<i>instance</i>	A FlexCAN instance number
<i>type</i>	The FlexCAN RX mask type

Definition at line 288 of file flexcan\_driver.c.

**16.50.5.24** void FLEXCAN\_DRV\_SetRxMb14Mask ( uint8\_t *instance*, flexcan\_msgbuff\_id\_type\_t *id\_type*, uint32\_t *mask* )

Sets the FlexCAN Rx MB 14 mask (standard or extended).

## Parameters

<i>instance</i>	A FlexCAN instance number
-----------------	---------------------------

<i>id_type</i>	Standard ID or extended ID
<i>mask</i>	Mask value

Definition at line 416 of file flexcan\_driver.c.

16.50.5.25 void FLEXCAN\_DRV\_SetRxMb15Mask ( uint8\_t *instance*, flexcan\_msgbuff\_id\_type\_t *id\_type*, uint32\_t *mask* )

Sets the FlexCAN Rx MB 15 mask (standard or extended).

**Parameters**

<i>instance</i>	A FlexCAN instance number
<i>id_type</i>	Standard ID or extended ID
<i>mask</i>	Mask value

Definition at line 460 of file flexcan\_driver.c.

16.50.5.26 void FLEXCAN\_DRV\_SetRxMbGlobalMask ( uint8\_t *instance*, flexcan\_msgbuff\_id\_type\_t *id\_type*, uint32\_t *mask* )

Sets the FlexCAN Rx MB global mask (standard or extended).

**Parameters**

<i>instance</i>	A FlexCAN instance number
<i>id_type</i>	Standard ID or extended ID
<i>mask</i>	Mask value

Definition at line 372 of file flexcan\_driver.c.

## 16.51 FlexIO Common Driver

### 16.51.1 Detailed Description

Common services for FlexIO drivers.

The Flexio Common driver layer contains services used by all Flexio drivers. The need for this layer derives from the requirement to allow multiple Flexio drivers to run in parallel on the same device, to the extent that enough hardware resources (shifters and timers) are available.

#### Functionality

The Flexio Common driver layer provides functions for device initialization and reset. Before using any Flexio driver the device must first be initialized using function [FLEXIO\\_DRV\\_InitDevice\(\)](#). Then any number of Flexio drivers can be initialized on the same device, to the extent that enough hardware resources (shifters and timers) are available. Driver initialization functions will return STATUS\_ERROR if not enough resources are available for a new driver.

#### Important Notes

Calling any Flexio common function will destroy any driver that is active on that device. Normally these functions should be called only when there are no active driver instances on the device.

#### Enumerations

- enum [flexio\\_driver\\_type\\_t](#) { [FLEXIO\\_DRIVER\\_TYPE\\_INTERRUPTS](#) = 0U, [FLEXIO\\_DRIVER\\_TYPE\\_POLLING](#) = 1U, [FLEXIO\\_DRIVER\\_TYPE\\_DMA](#) = 2U }

*Driver type: interrupts/polling/DMA Implements : flexio\_driver\_type\_t Class.*

#### FLEXIO\_I2C Driver

- status\_t [FLEXIO\\_DRV\\_InitDevice](#) (uint32\_t instance, flexio\_device\_state\_t \*deviceState)  
*Initializes the FlexIO device.*
- status\_t [FLEXIO\\_DRV\\_DeinitDevice](#) (uint32\_t instance)  
*De-initializes the FlexIO device.*
- status\_t [FLEXIO\\_DRV\\_Reset](#) (uint32\_t instance)  
*Resets the FlexIO device.*

### 16.51.2 Enumeration Type Documentation

#### 16.51.2.1 enum flexio\_driver\_type\_t

Driver type: interrupts/polling/DMA Implements : flexio\_driver\_type\_t Class.

#### Enumerator

**FLEXIO\_DRIVER\_TYPE\_INTERRUPTS** Driver uses interrupts for data transfers

**FLEXIO\_DRIVER\_TYPE\_POLLING** Driver is based on polling

**FLEXIO\_DRIVER\_TYPE\_DMA** Driver uses DMA for data transfers

Definition at line 46 of file flexio.h.

### 16.51.3 Function Documentation

#### 16.51.3.1 status\_t FLEXIO\_DRV\_DeinitDevice ( uint32\_t instance )

De-initializes the FlexIO device.

This function de-initializes the FlexIO device.

**Parameters**

<i>instance</i>	FLEXIO peripheral instance number
-----------------	-----------------------------------

**Returns**

Error or success status returned by API

Definition at line 125 of file flexio\_common.c.

**16.51.3.2** `status_t FLEXIO_DRV_InitDevice ( uint32_t instance, flexio_device_state_t * deviceState )`

Initializes the FlexIO device.

This function resets the FlexIO device, enables interrupts in interrupt manager and enables the device.

**Parameters**

<i>instance</i>	FLEXIO peripheral instance number
<i>deviceState</i>	Pointer to the FLEXIO device context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the device is de-initialized using <a href="#">FLEXIO_DRV_DeinitDevice()</a> .

**Returns**

Error or success status returned by API

Definition at line 86 of file flexio\_common.c.

**16.51.3.3** `status_t FLEXIO_DRV_Reset ( uint32_t instance )`

Resets the FlexIO device.

This function resets the FlexIO device.

**Parameters**

<i>instance</i>	FLEXIO peripheral instance number
-----------------	-----------------------------------

**Returns**

Error or success status returned by API

Definition at line 150 of file flexio\_common.c.

## 16.52 FlexIO I2C Driver

### 16.52.1 Detailed Description

I2C communication over FlexIO module (FLEXIO\_I2C)

The FLEXIO\_I2C Driver allows communication on an I2C bus using the FlexIO module in the S32K1xx processors.

#### Features

- Master operation only
- Interrupt, DMA or polling mode
- Provides blocking and non-blocking transmit and receive functions
- 7-bit addressing
- Clock stretching
- Configurable baud rate

#### Functionality

Before using any Flexio driver the device must first be initialized using function `FLEXIO_DRV_InitDevice`. Then the FLEXIO\_I2C Driver must be initialized using functions `FLEXIO_I2C_DRV_MasterInit()`. It is possible to use more driver instances on the same FlexIO device, as long as sufficient resources are available. Different driver instances on the same FlexIO device can function independently of each other. When it is no longer needed, the driver can be de-initialized, using `FLEXIO_I2C_DRV_MasterDeinit()`. This will release the hardware resources, allowing other driver instances to be initialized.

#### Master Mode

Master Mode provides functions for transmitting or receiving data to/from any I2C slave. Slave address and baud rate are provided at initialization time through the master configuration structure, but they can be changed at runtime by using `FLEXIO_I2C_DRV_MasterSetBaudRate()` or `FLEXIO_I2C_DRV_MasterSetSlaveAddr()`. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call `FLEXIO_I2C_DRV_MasterGetBaudRate()` after `FLEXIO_I2C_DRV_MasterSetBaudRate()` to check what baud rate was actually set.

To send or receive data to/from the currently configured slave address, use functions `FLEXIO_I2C_DRV_MasterSendData()` or `FLEXIO_I2C_DRV_MasterReceiveData()` (or their blocking counterparts). Parameter `sendStop` can be used to chain multiple transfers with repeated START condition between them, for example when sending a command and then immediately receiving a response. The application should ensure that any send or receive transfer with `sendStop` set to `false` is followed by another transfer. The last transfer from a chain should always have `sendStop` set to `true`. This driver does not support continuous send/receive using a user callback function. The callback function is only used to signal the end of a transfer.

Blocking operations will return only when the transfer is completed, either successfully or with error. Non-blocking operations will initiate the transfer and return `STATUS_SUCCESS`, but the module is still busy with the transfer and another transfer can't be initiated until the current transfer is complete. The application will be notified through the user callback when the transfer completes, or it can check the status of the current transfer by calling `FLEXIO_I2C_DRV_MasterGetStatus()`. If the transfer is still ongoing this function will return `STATUS_BUSY`. If the transfer is completed, the function will return either `STATUS_SUCCESS` or an error code, depending on the outcome of the last transfer.

The driver supports interrupt, DMA and polling mode. In polling mode the function `FLEXIO_I2C_DRV_MasterGetStatus()` ensures the progress of the transfer by checking and handling transmit and receive events reported by the FlexIO module. The application should ensure that this function is called often enough (at least once per transferred byte) to avoid Tx underflows or Rx overflows. In DMA mode the DMA channels that will be used by the driver are received through the configuration structure. The channels must be initialized by the application before the flexio\_i2c driver is initialized. The flexio\_i2c driver will only set the DMA request source.



### Important Notes

- There is one limitation of flexio\_i2c which no Stop condition is generated when aborting a transfer due to NACK reception.
- Before using the FLEXIO\_I2C Driver the FlexIO clock must be configured. Refer to Clock Manager for clock configuration.
- Before using the FLEXIO\_I2C Driver the pins must be routed to the FlexIO module. Refer to PINS Driver for pin routing configuration. Note that any of the available FlexIO pins can be used for SDA and SCL (configurable at initialization time).
- The driver enables the interrupts for the corresponding FlexIO module, but any interrupt priority setting must be done by the application.
- Aborting a transfer with the function `FLEXIO_I2C_DRV_MasterTransferAbort()` can't generally be done safely due to device limitation; there is no way to know the exact stage of the transfer, and if we disable the module during the ACK bit (transmit) or during a 0 data bit (receive) the slave will hold the SDA line low forever and block the I2C bus. Therefore this function should only be used in extreme circumstances, and the application must have a way to reset the I2C slave. NACK reception is the only exception, as there is no slave to hold the line low, so in this case the driver will automatically abort the transfer.
- The module can handle clock stretching done by the slave, but will not do clock stretching when the application does not provide data fast enough, so Tx underflows and Rx overflows are possible. This can be an issue especially in polling mode if the function `FLEXIO_I2C_DRV_MasterGetStatus()` is not called often enough.
- Due to device limitations it is not always possible to tell the difference between NACK reception and receiver overflow. When in doubt, the driver will treat these events as overflow and continue the transfer, in order to avoid the risk of blocking the i2c bus.
- The driver does not support multi-master mode. It does not detect arbitration loss condition.
- Timeout feature for blocking transfers does not work in polling mode.
- This driver needs two shifters and two timers for its operation. Initialization will fail if there are not enough shifters and timers available on the FlexIO device.
- This driver needs two DMA channels for its operation when it is initialized in DMA mode. The DMA channels must be initialized by the application before initializing the driver. Refer to EDMA driver for DMA channels initialization.
- If the application uses an RTOS, this driver uses a semaphore for blocking transfers. Initialization will fail if the semaphore cannot be created. If the driver uses polling mode no semaphore is used.
- If the application uses an RTOS, the FlexIO drivers use a mutex for channel allocation. Only one mutex per device is needed, not per driver instance. Device initialization will fail if the mutex cannot be created.

### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\drivers\src\flexio\flexio_common.c
${S32SDK_PATH}\platform\drivers\src\flexio\flexio_i2c_driver.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc
${S32SDK_PATH}\platform\drivers\src\flexio
```

## Compile symbols

No special symbols are required for this component

## Dependencies

[Clock Manager OS Interface \(OSIF\)](#) [Interrupt Manager \(Interrupt\)](#) [Enhanced Direct Memory Access \(eDMA\)](#)

## Data Structures

- struct [flexio\\_i2c\\_master\\_user\\_config\\_t](#)  
*Master configuration structure. [More...](#)*
- struct [flexio\\_i2c\\_master\\_state\\_t](#)  
*Master internal context structure. [More...](#)*

## FLEXIO\_I2C Driver

- status\_t [FLEXIO\\_I2C\\_DRV\\_MasterInit](#) (uint32\_t instance, const [flexio\\_i2c\\_master\\_user\\_config\\_t](#) \*userConfigPtr, [flexio\\_i2c\\_master\\_state\\_t](#) \*master)  
*Initialize the FLEXIO\_I2C master mode driver.*
- status\_t [FLEXIO\\_I2C\\_DRV\\_MasterDeinit](#) ([flexio\\_i2c\\_master\\_state\\_t](#) \*master)  
*De-initialize the FLEXIO\_I2C master mode driver.*
- status\_t [FLEXIO\\_I2C\\_DRV\\_MasterSetBaudRate](#) ([flexio\\_i2c\\_master\\_state\\_t](#) \*master, uint32\_t baudRate)  
*Set the baud rate for any subsequent I2C communication.*
- status\_t [FLEXIO\\_I2C\\_DRV\\_MasterGetBaudRate](#) ([flexio\\_i2c\\_master\\_state\\_t](#) \*master, uint32\_t \*baudRate)  
*Get the currently configured baud rate.*
- status\_t [FLEXIO\\_I2C\\_DRV\\_MasterSetSlaveAddr](#) ([flexio\\_i2c\\_master\\_state\\_t](#) \*master, const uint16\_t address)  
*Set the slave address for any subsequent I2C communication.*
- status\_t [FLEXIO\\_I2C\\_DRV\\_MasterSendData](#) ([flexio\\_i2c\\_master\\_state\\_t](#) \*master, const uint8\_t \*txBuff, uint32\_t txSize, bool sendStop)  
*Perform a non-blocking send transaction on the I2C bus.*
- status\_t [FLEXIO\\_I2C\\_DRV\\_MasterSendDataBlocking](#) ([flexio\\_i2c\\_master\\_state\\_t](#) \*master, const uint8\_t \*txBuff, uint32\_t txSize, bool sendStop, uint32\_t timeout)  
*Perform a blocking send transaction on the I2C bus.*
- status\_t [FLEXIO\\_I2C\\_DRV\\_MasterReceiveData](#) ([flexio\\_i2c\\_master\\_state\\_t](#) \*master, uint8\_t \*rxBuff, uint32\_t rxSize, bool sendStop)  
*Perform a non-blocking receive transaction on the I2C bus.*
- status\_t [FLEXIO\\_I2C\\_DRV\\_MasterReceiveDataBlocking](#) ([flexio\\_i2c\\_master\\_state\\_t](#) \*master, uint8\_t \*rxBuff, uint32\_t rxSize, bool sendStop, uint32\_t timeout)  
*Perform a blocking receive transaction on the I2C bus.*
- status\_t [FLEXIO\\_I2C\\_DRV\\_MasterTransferAbort](#) ([flexio\\_i2c\\_master\\_state\\_t](#) \*master)  
*Aborts a non-blocking I2C master transaction.*
- status\_t [FLEXIO\\_I2C\\_DRV\\_MasterGetStatus](#) ([flexio\\_i2c\\_master\\_state\\_t](#) \*master, uint32\_t \*bytesRemaining)  
*Get the status of the current non-blocking I2C master transaction.*
- void [FLEXIO\\_I2C\\_DRV\\_GetDefaultConfig](#) ([flexio\\_i2c\\_master\\_user\\_config\\_t](#) \*userConfigPtr)  
*Returns default configuration structure for FLEXIO\_I2C.*
- status\_t [FLEXIO\\_I2C\\_DRV\\_GenerateNineClock](#) ([flexio\\_i2c\\_master\\_state\\_t](#) \*master)  
*Generate nine clock on SCL line to free SDA line.*
- status\_t [FLEXIO\\_I2C\\_DRV\\_StatusGenerateNineClock](#) ([flexio\\_i2c\\_master\\_state\\_t](#) \*master)  
*Indicate the generation nine clock is done or not.*
- bool [FLEXIO\\_I2C\\_DRV\\_GetBusStatus](#) (const [flexio\\_i2c\\_master\\_state\\_t](#) \*master, bool sdaLine)  
*Check status whether SDA or SCL line be low or high.*

### 16.52.2 Data Structure Documentation

#### 16.52.2.1 struct flexio\_i2c\_master\_user\_config\_t

Master configuration structure.

This structure is used to provide configuration parameters for the flexio\_i2c master at initialization time. Implements : flexio\_i2c\_master\_user\_config\_t\_Class

Definition at line 84 of file flexio\_i2c\_driver.h.

##### Data Fields

- uint16\_t [slaveAddress](#)
- [flexio\\_driver\\_type\\_t](#) [driverType](#)
- uint32\_t [baudRate](#)
- uint8\_t [sdaPin](#)
- uint8\_t [sclPin](#)
- i2c\_master\_callback\_t [callback](#)
- void \* [callbackParam](#)
- uint8\_t [rxDMAChannel](#)
- uint8\_t [txDMAChannel](#)

##### Field Documentation

#### 16.52.2.1.1 uint32\_t baudRate

Baud rate in hertz

Definition at line 88 of file flexio\_i2c\_driver.h.

#### 16.52.2.1.2 i2c\_master\_callback\_t callback

User callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if it is not needed

Definition at line 91 of file flexio\_i2c\_driver.h.

#### 16.52.2.1.3 void\* callbackParam

Parameter for the callback function

Definition at line 95 of file flexio\_i2c\_driver.h.

#### 16.52.2.1.4 flexio\_driver\_type\_t driverType

Driver type: interrupts/polling/DMA

Definition at line 87 of file flexio\_i2c\_driver.h.

#### 16.52.2.1.5 uint8\_t rxDMAChannel

Rx DMA channel number. Only used in DMA mode

Definition at line 96 of file flexio\_i2c\_driver.h.

#### 16.52.2.1.6 uint8\_t sclPin

Flexio pin to use as I2C SCL pin

Definition at line 90 of file flexio\_i2c\_driver.h.

#### 16.52.2.1.7 uint8\_t sdaPin

Flexio pin to use as I2C SDA pin

Definition at line 89 of file flexio\_i2c\_driver.h.

#### 16.52.2.1.8 uint16\_t slaveAddress

Slave address, 7-bit

Definition at line 86 of file flexio\_i2c\_driver.h.

#### 16.52.2.1.9 uint8\_t txDMAChannel

Tx DMA channel number. Only used in DMA mode

Definition at line 97 of file flexio\_i2c\_driver.h.

#### 16.52.2.2 struct flexio\_i2c\_master\_state\_t

Master internal context structure.

This structure is used by the driver for its internal logic. It must be provided by the application through the [FLEXIO\\_I2C\\_DRV\\_MasterInit\(\)](#) function, then it cannot be freed until the driver is de-initialized using [FLEXIO\\_I2C\\_DRV\\_MasterDeinit\(\)](#). The application should make no assumptions about the content of this structure.

Definition at line 109 of file flexio\_i2c\_driver.h.

### 16.52.3 Function Documentation

#### 16.52.3.1 status\_t FLEXIO\_I2C\_DRV\_GenerateNineClock ( flexio\_i2c\_master\_state\_t \* master )

Generate nine clock on SCL line to free SDA line.

This function should be called when SDA line be stuck in low.

##### Parameters

<i>master</i>	Pointer to the FLEXIO_I2C master driver context structure.
---------------	--

##### Returns

STATUS\_BUSY: Driver is transferring data, STATUS\_SUCCESS: Function started generating clock

Definition at line 1681 of file flexio\_i2c\_driver.c.

#### 16.52.3.2 bool FLEXIO\_I2C\_DRV\_GetBusStatus ( const flexio\_i2c\_master\_state\_t \* master, bool sdaLine )

Check status whether SDA or SCL line be low or high.

##### Parameters

<i>master</i>	Pointer to the FLEXIO_I2C master driver context structure.
<i>sdaLine</i>	true - function return status of SDA line. false - function return status of SCL line.

##### Returns

true: Pin selected is high, false: Pin selected is low

Definition at line 1766 of file flexio\_i2c\_driver.c.

#### 16.52.3.3 void FLEXIO\_I2C\_DRV\_GetDefaultConfig ( flexio\_i2c\_master\_user\_config\_t \* userConfigPtr )

Returns default configuration structure for FLEXIO\_I2C.

## Parameters

<i>userConfigPtr</i>	Pointer to the FLEXIO_I2C user configuration structure.
----------------------	---

Definition at line 1658 of file flexio\_i2c\_driver.c.

**16.52.3.4** `status_t FLEXIO_I2C_DRV_MasterDeinit ( flexio_i2c_master_state_t * master )`

De-initialize the FLEXIO\_I2C master mode driver.

This function de-initializes the FLEXIO\_I2C driver in master mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

## Parameters

<i>master</i>	Pointer to the FLEXIO_I2C master driver context structure.
---------------	--

## Returns

Error or success status returned by API

Definition at line 1286 of file flexio\_i2c\_driver.c.

**16.52.3.5** `status_t FLEXIO_I2C_DRV_MasterGetBaudRate ( flexio_i2c_master_state_t * master, uint32_t * baudRate )`

Get the currently configured baud rate.

This function returns the currently configured I2C baud rate.

## Parameters

<i>master</i>	Pointer to the FLEXIO_I2C master driver context structure.
<i>baudRate</i>	the current baud rate in hertz

## Returns

Error or success status returned by API

Definition at line 1356 of file flexio\_i2c\_driver.c.

**16.52.3.6** `status_t FLEXIO_I2C_DRV_MasterGetStatus ( flexio_i2c_master_state_t * master, uint32_t * bytesRemaining )`

Get the status of the current non-blocking I2C master transaction.

This function returns the current status of a non-blocking I2C master transaction. A return code of STATUS\_BUSY means the transfer is still in progress. Otherwise the function returns a status reflecting the outcome of the last transfer. When the driver is initialized in polling mode this function also advances the transfer by checking and handling the transmit and receive events, so it must be called frequently to avoid overflows or underflows.

## Parameters

<i>master</i>	Pointer to the FLEXIO_I2C master driver context structure.
<i>bytesRemaining</i>	The remaining number of bytes to be transferred

## Returns

Error or success status returned by API

Definition at line 1603 of file flexio\_i2c\_driver.c.

**16.52.3.7** `status_t FLEXIO_I2C_DRV_MasterInit ( uint32_t instance, const flexio_i2c_master_user_config_t * userConfigPtr, flexio_i2c_master_state_t * master )`

Initialize the FLEXIO\_I2C master mode driver.

This function initializes the FLEXIO\_I2C driver in master mode.

## Parameters

<i>instance</i>	FLEXIO peripheral instance number
<i>userConfigPtr</i>	Pointer to the FLEXIO_I2C master user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
<i>master</i>	Pointer to the FLEXIO_I2C master driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using <a href="#">FLEXIO_I2C_DRV_MasterDeinit()</a> .

## Returns

Error or success status returned by API

Definition at line 1195 of file flexio\_i2c\_driver.c.

**16.52.3.8** `status_t FLEXIO_I2C_DRV_MasterReceiveData ( flexio_i2c_master_state_t * master, uint8_t * rxBuff, uint32_t rxSize, bool sendStop )`

Perform a non-blocking receive transaction on the I2C bus.

This function starts the reception of a block of data from the currently configured slave address and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the [FLEXIO\\_I2C\\_DRV\\_MasterGetStatus](#) function (if the driver is initialized in polling mode). Use [FLEXIO\\_I2C\\_DRV\\_MasterGetStatus\(\)](#) to check the progress of the reception.

## Parameters

<i>master</i>	Pointer to the FLEXIO_I2C master driver context structure.
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred
<i>sendStop</i>	specifies whether or not to generate stop condition after the reception

## Returns

Error or success status returned by API

Definition at line 1494 of file flexio\_i2c\_driver.c.

**16.52.3.9** `status_t FLEXIO_I2C_DRV_MasterReceiveDataBlocking ( flexio_i2c_master_state_t * master, uint8_t * rxBuff, uint32_t rxSize, bool sendStop, uint32_t timeout )`

Perform a blocking receive transaction on the I2C bus.

This function receives a block of data from the currently configured slave address, and only returns when the transmission is complete.

## Parameters

<i>master</i>	Pointer to the FLEXIO_I2C master driver context structure.
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred
<i>sendStop</i>	specifies whether or not to generate stop condition after the reception
<i>timeout</i>	timeout for the transfer in milliseconds

## Returns

Error or success status returned by API

Definition at line 1523 of file flexio\_i2c\_driver.c.

**16.52.3.10** `status_t FLEXIO_I2C_DRV_MasterSendData ( flexio_i2c_master_state_t * master, const uint8_t * txBuff, uint32_t txSize, bool sendStop )`

Perform a non-blocking send transaction on the I2C bus.

This function starts the transmission of a block of data to the currently configured slave address and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the `FLEXIO_I2C_DRV_MasterGetStatus` function (if the driver is initialized in polling mode). Use `FLEXIO_I2C_DRV_MasterGetStatus()` to check the progress of the transmission.

#### Parameters

<i>master</i>	Pointer to the FLEXIO_I2C master driver context structure.
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred
<i>sendStop</i>	specifies whether or not to generate stop condition after the transmission

#### Returns

Error or success status returned by API

Definition at line 1417 of file `flexio_i2c_driver.c`.

**16.52.3.11** `status_t FLEXIO_I2C_DRV_MasterSendDataBlocking ( flexio_i2c_master_state_t * master, const uint8_t * txBuff, uint32_t txSize, bool sendStop, uint32_t timeout )`

Perform a blocking send transaction on the I2C bus.

This function sends a block of data to the currently configured slave address, and only returns when the transmission is complete.

#### Parameters

<i>master</i>	Pointer to the FLEXIO_I2C master driver context structure.
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred
<i>sendStop</i>	specifies whether or not to generate stop condition after the transmission
<i>timeout</i>	timeout for the transfer in milliseconds

#### Returns

Error or success status returned by API

Definition at line 1446 of file `flexio_i2c_driver.c`.

**16.52.3.12** `status_t FLEXIO_I2C_DRV_MasterSetBaudRate ( flexio_i2c_master_state_t * master, uint32_t baudRate )`

Set the baud rate for any subsequent I2C communication.

This function sets the baud rate (SCL frequency) for the I2C master. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call `FLEXIO_I2C_DRV_MasterGetBaudRate()` after `FLEXIO_I2C_DRV_MasterSetBaudRate()` to check what baud rate was actually set.

#### Parameters

<i>master</i>	Pointer to the FLEXIO_I2C master driver context structure.
---------------	--

<i>baudRate</i>	the desired baud rate in hertz
-----------------	--------------------------------

**Returns**

Error or success status returned by API

Definition at line 1312 of file flexio\_i2c\_driver.c.

**16.52.3.13** `status_t FLEXIO_I2C_DRV_MasterSetSlaveAddr ( flexio_i2c_master_state_t * master, const uint16_t address )`

Set the slave address for any subsequent I2C communication.

This function sets the slave address which will be used for any future transfer.

**Parameters**

<i>master</i>	Pointer to the FLEXIO_I2C master driver context structure.
<i>address</i>	slave address, 7-bit

**Returns**

Error or success status returned by API

Definition at line 1395 of file flexio\_i2c\_driver.c.

**16.52.3.14** `status_t FLEXIO_I2C_DRV_MasterTransferAbort ( flexio_i2c_master_state_t * master )`

Aborts a non-blocking I2C master transaction.

This function aborts a non-blocking I2C transfer.

**Parameters**

<i>master</i>	Pointer to the FLEXIO_I2C master driver context structure.
---------------	--

**Returns**

Error or success status returned by API

Definition at line 1571 of file flexio\_i2c\_driver.c.

**16.52.3.15** `status_t FLEXIO_I2C_DRV_StatusGenerateNineClock ( flexio_i2c_master_state_t * master )`

Indicate the generation nine clock is done or not.

**Parameters**

<i>master</i>	Pointer to the FLEXIO_I2C master driver context structure.
---------------	--

**Returns**

STATUS\_BUSY: Clock generation not done yet, STATUS\_SUCCESS: Device finished generating nine clock

Definition at line 1730 of file flexio\_i2c\_driver.c.



## 16.53 FlexIO I2S Driver

### 16.53.1 Detailed Description

I2S communication over FlexIO module (FLEXIO\_I2S)

The FLEXIO\_I2S Driver allows communication on an I2S bus using the FlexIO module in the S32K1xx processors.

#### Features

- Master or slave operation
- Interrupt, DMA or polling mode
- Provides blocking and non-blocking transmit and receive functions
- Configurable baud rate and bit count

#### Functionality

Before using any Flexio driver the device must first be initialized using function `FLEXIO_DRV_InitDevice`. Then the FLEXIO\_I2S Driver must be initialized, using functions `FLEXIO_I2S_DRV_MasterInit()` or `FLEXIO_I2S_DRV_SlaveInit()`. It is possible to use more driver instances on the same FlexIO device, as long as sufficient resources are available. Different driver instances on the same FlexIO device can function independently of each other. When it is no longer needed, the driver can be de-initialized, using `FLEXIO_I2S_DRV_MasterDeinit()` or `FLEXIO_I2S_DRV_SlaveDeinit()`. This will release the hardware resources, allowing other driver instances to be initialized.

#### Master Mode

Master Mode provides functions for transmitting or receiving data to/from any I2S slave. The number of bits per word and the baud rate are provided at initialization time through the master configuration structure, but they can be changed at runtime by using `FLEXIO_I2S_DRV_MasterSetConfig()`. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call `FLEXIO_I2S_DRV_MasterGetBaudRate()` to check what baud rate was actually set.

To send or receive data to/from the currently configured slave address, use functions `FLEXIO_I2S_DRV_MasterSendData()` or `FLEXIO_I2S_DRV_MasterReceiveData()` (or their blocking counterparts). The driver is not full-duplex, only one direction (send or receive) can be used at one time. It is possible to configure both Rx and Tx pin to use the same Flexio pin.

Continuous send/receive can be realized by registering a user callback function. When the driver completes the transmission or reception of the current buffer, it will invoke the user callback with an appropriate event. The callback function can use `FLEXIO_I2S_DRV_MasterSetTxBuffer()` or `FLEXIO_I2S_DRV_MasterSetRxBuffer()` to provide a new buffer.

Blocking operations will return only when the transfer is completed, either successfully or with error. Non-blocking operations will initiate the transfer and return `STATUS_SUCCESS`, but the module is still busy with the transfer and another transfer can't be initiated until the current transfer is complete. The application will be notified through the user callback when the transfer completes, or it can check the status of the current transfer by calling `FLEXIO_I2S_DRV_MasterGetStatus()`. If the transfer is still ongoing this function will return `STATUS_BUSY`. If the transfer is completed, the function will return either `STATUS_SUCCESS` or an error code, depending on the outcome of the last transfer.

The driver supports interrupt, DMA and polling mode. In polling mode the function `FLEXIO_I2S_DRV_MasterGetStatus()` ensures the progress of the transfer by checking and handling transmit and receive events reported by the FlexIO module. The application should ensure that this function is called often enough (at least once per transferred byte) to avoid Tx underflows or Rx overflows. In DMA mode the DMA channels that will be used by the driver are received through the configuration structure. The channels must be initialized by the application before the flexio\_i2s driver is initialized. The flexio\_i2s driver will only set the DMA request source.

## Slave Mode

Slave Mode is very similar to master mode, the main difference being that the [FLEXIO\\_I2S\\_DRV\\_SlaveInit\(\)](#) function initializes the FlexIO module to use the clock signal received from the master instead of generating it. Consequently, there is no baud rate setting in slave mode. Other than that, the slave mode offers a similar interface to the master mode. [FLEXIO\\_I2S\\_DRV\\_MasterSendData\(\)](#) or [FLEXIO\\_I2S\\_DRV\\_MasterReceiveData\(\)](#) (or their blocking counterparts) can be used to initiate transfers, and [FLEXIO\\_I2S\\_DRV\\_SlaveGetStatus\(\)](#) is used to check the status of the transfer and advance the transfer in polling mode. All other specifications from the Master Mode description apply for Slave Mode too.

## Important Notes

- Before using the FLEXIO\_I2S Driver the FlexIO clock must be configured. Refer to Clock Manager for clock configuration.
- Before using the FLEXIO\_I2S Driver the pins must be routed to the FlexIO module. Refer to PINS Driver for pin routing configuration. Note that any of the available FlexIO pins can be used for any of the TX, RX, SCK and WS signals (configurable at initialization time). If more than one driver instance is used on the same Flexio module, it is the responsibility of the application to ensure there are no conflicts between pins.
- The driver enables the interrupts for the corresponding FlexIO module, but any interrupt priority setting must be done by the application.
- Timeout feature for blocking transfers does not work in polling mode.
- This driver needs two shifters and two timers for its operation. Initialization will fail if there are not enough shifters and timers available on the FlexIO device.
- This driver needs two DMA channels for its operation when it is initialized in DMA mode. The DMA channels must be initialized by the application before initializing the driver. Refer to EDMA driver for DMA channels initialization.
- If the application uses an RTOS, this driver uses a semaphore for blocking transfers. Initialization will fail if the semaphore cannot be created. If the driver uses polling mode no semaphore is used.
- If the application uses an RTOS, the FlexIO drivers use a mutex for channel allocation. Only one mutex per device is needed, not per driver instance. Device initialization will fail if the mutex cannot be created.
- For transfers where the data size is more than 1 byte (bitsWidth is greater than 8) the driver assumes that the data buffers are defined with the proper type (uint16\_t or uint32\_t) and are properly aligned.

## Integration guideline

### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\drivers\src\flexio\flexio_common.c
${S32SDK_PATH}\platform\drivers\src\flexio\flexio_i2s_driver.c
```

### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc
${S32SDK_PATH}\platform\drivers\src\flexio
```

### Compile symbols

No special symbols are required for this component

### Dependencies

[Clock Manager OS Interface \(OSIF\)](#) [Interrupt Manager \(Interrupt\)](#) [Enhanced Direct Memory Access \(eDMA\)](#)

## Data Structures

- struct [flexio\\_i2s\\_master\\_user\\_config\\_t](#)  
Master configuration structure. [More...](#)
- struct [flexio\\_i2s\\_slave\\_user\\_config\\_t](#)  
Slave configuration structure. [More...](#)
- struct [flexio\\_i2s\\_master\\_state\\_t](#)  
Master internal context structure. [More...](#)

## Typedefs

- typedef [flexio\\_i2s\\_master\\_state\\_t](#) [flexio\\_i2s\\_slave\\_state\\_t](#)  
Slave internal context structure.

## FLEXIO\_I2S Driver

- status\_t [FLEXIO\\_I2S\\_DRV\\_MasterInit](#) (uint32\_t instance, const [flexio\\_i2s\\_master\\_user\\_config\\_t](#) \*userConfigPtr, [flexio\\_i2s\\_master\\_state\\_t](#) \*master)  
Initialize the FLEXIO\_I2S master mode driver.
- status\_t [FLEXIO\\_I2S\\_DRV\\_MasterDeinit](#) ([flexio\\_i2s\\_master\\_state\\_t](#) \*master)  
De-initialize the FLEXIO\_I2S master mode driver.
- status\_t [FLEXIO\\_I2S\\_DRV\\_MasterSetConfig](#) ([flexio\\_i2s\\_master\\_state\\_t](#) \*master, uint32\_t baudRate, uint8\_t bitsWidth)  
Set the baud rate and bit width for any subsequent I2S communication.
- status\_t [FLEXIO\\_I2S\\_DRV\\_MasterGetBaudRate](#) ([flexio\\_i2s\\_master\\_state\\_t](#) \*master, uint32\_t \*baudRate)  
Get the currently configured baud rate.
- status\_t [FLEXIO\\_I2S\\_DRV\\_MasterSendData](#) ([flexio\\_i2s\\_master\\_state\\_t](#) \*master, const uint8\_t \*txBuff, uint32\_t txSize)  
Perform a non-blocking send transaction on the I2S bus.
- status\_t [FLEXIO\\_I2S\\_DRV\\_MasterSendDataBlocking](#) ([flexio\\_i2s\\_master\\_state\\_t](#) \*master, const uint8\_t \*txBuff, uint32\_t txSize, uint32\_t timeout)  
Perform a blocking send transaction on the I2S bus.
- status\_t [FLEXIO\\_I2S\\_DRV\\_MasterReceiveData](#) ([flexio\\_i2s\\_master\\_state\\_t](#) \*master, uint8\_t \*rxBuff, uint32\_t rxSize)  
Perform a non-blocking receive transaction on the I2S bus.
- status\_t [FLEXIO\\_I2S\\_DRV\\_MasterReceiveDataBlocking](#) ([flexio\\_i2s\\_master\\_state\\_t](#) \*master, uint8\_t \*rxBuff, uint32\_t rxSize, uint32\_t timeout)  
Perform a blocking receive transaction on the I2S bus.
- status\_t [FLEXIO\\_I2S\\_DRV\\_MasterTransferAbort](#) ([flexio\\_i2s\\_master\\_state\\_t](#) \*master)  
Aborts a non-blocking I2S master transaction.
- status\_t [FLEXIO\\_I2S\\_DRV\\_MasterGetStatus](#) ([flexio\\_i2s\\_master\\_state\\_t](#) \*master, uint32\_t \*bytesRemaining)  
Get the status of the current non-blocking I2S master transaction.
- status\_t [FLEXIO\\_I2S\\_DRV\\_MasterSetRxBuffer](#) ([flexio\\_i2s\\_master\\_state\\_t](#) \*master, uint8\_t \*rxBuff, uint32\_t rxSize)  
Provide a buffer for receiving data.
- status\_t [FLEXIO\\_I2S\\_DRV\\_MasterSetTxBuffer](#) ([flexio\\_i2s\\_master\\_state\\_t](#) \*master, const uint8\_t \*txBuff, uint32\_t txSize)  
Provide a buffer for transmitting data.
- status\_t [FLEXIO\\_I2S\\_DRV\\_Slavelnit](#) (uint32\_t instance, const [flexio\\_i2s\\_slave\\_user\\_config\\_t](#) \*userConfigPtr, [flexio\\_i2s\\_slave\\_state\\_t](#) \*slave)  
Initialize the FLEXIO\_I2S slave mode driver.

- status\_t [FLEXIO\\_I2S\\_DRV\\_SlaveDeinit](#) ([flexio\\_i2s\\_slave\\_state\\_t](#) \*slave)  
*De-initialize the FLEXIO\_I2S slave mode driver.*
- status\_t [FLEXIO\\_I2S\\_DRV\\_SlaveSetConfig](#) ([flexio\\_i2s\\_slave\\_state\\_t](#) \*slave, uint8\_t bitsWidth)  
*Set the bit width for any subsequent I2S communication.*
- status\_t [FLEXIO\\_I2S\\_DRV\\_SlaveSendData](#) ([flexio\\_i2s\\_slave\\_state\\_t](#) \*slave, const uint8\_t \*txBuff, uint32\_t txSize)  
*Perform a non-blocking send transaction on the I2S bus.*
- status\_t [FLEXIO\\_I2S\\_DRV\\_SlaveSendDataBlocking](#) ([flexio\\_i2s\\_slave\\_state\\_t](#) \*slave, const uint8\_t \*txBuff, uint32\_t txSize, uint32\_t timeout)  
*Perform a blocking send transaction on the I2S bus.*
- status\_t [FLEXIO\\_I2S\\_DRV\\_SlaveReceiveData](#) ([flexio\\_i2s\\_slave\\_state\\_t](#) \*slave, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Perform a non-blocking receive transaction on the I2S bus.*
- status\_t [FLEXIO\\_I2S\\_DRV\\_SlaveReceiveDataBlocking](#) ([flexio\\_i2s\\_slave\\_state\\_t](#) \*slave, uint8\_t \*rxBuff, uint32\_t rxSize, uint32\_t timeout)  
*Perform a blocking receive transaction on the I2S bus.*
- status\_t [FLEXIO\\_I2S\\_DRV\\_SlaveTransferAbort](#) ([flexio\\_i2s\\_slave\\_state\\_t](#) \*slave)  
*Aborts a non-blocking I2S slave transaction.*
- status\_t [FLEXIO\\_I2S\\_DRV\\_SlaveGetStatus](#) ([flexio\\_i2s\\_slave\\_state\\_t](#) \*slave, uint32\_t \*bytesRemaining)  
*Get the status of the current non-blocking I2S slave transaction.*
- status\_t [FLEXIO\\_I2S\\_DRV\\_SlaveSetRxBuffer](#) ([flexio\\_i2s\\_slave\\_state\\_t](#) \*slave, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Provide a buffer for receiving data.*
- status\_t [FLEXIO\\_I2S\\_DRV\\_SlaveSetTxBuffer](#) ([flexio\\_i2s\\_slave\\_state\\_t](#) \*slave, const uint8\_t \*txBuff, uint32\_t txSize)  
*Provide a buffer for transmitting data.*
- void [FLEXIO\\_I2S\\_DRV\\_MasterGetDefaultConfig](#) ([flexio\\_i2s\\_master\\_user\\_config\\_t](#) \*userConfigPtr)  
*Returns default configuration structure for FLEXIO\_I2S master.*
- void [FLEXIO\\_I2S\\_DRV\\_SlaveGetDefaultConfig](#) ([flexio\\_i2s\\_slave\\_user\\_config\\_t](#) \*userConfigPtr)  
*Returns default configuration structure for FLEXIO\_I2S slave.*

## 16.53.2 Data Structure Documentation

### 16.53.2.1 struct flexio\_i2s\_master\_user\_config\_t

Master configuration structure.

This structure is used to provide configuration parameters for the flexio\_i2s master at initialization time. Implements : flexio\_i2s\_master\_user\_config\_t\_Class

Definition at line 67 of file flexio\_i2s\_driver.h.

#### Data Fields

- [flexio\\_driver\\_type\\_t](#) driverType
- uint32\_t baudRate
- uint8\_t bitsWidth
- uint8\_t txPin
- uint8\_t rxPin
- uint8\_t sckPin
- uint8\_t wsPin
- i2s\_callback\_t callback
- void \* callbackParam
- uint8\_t rxDMAChannel
- uint8\_t txDMAChannel

## Field Documentation

### 16.53.2.1.1 uint32\_t baudRate

Baud rate in hertz

Definition at line 70 of file flexio\_i2s\_driver.h.

### 16.53.2.1.2 uint8\_t bitsWidth

Number of bits in a word - multiple of 8

Definition at line 71 of file flexio\_i2s\_driver.h.

### 16.53.2.1.3 i2s\_callback\_t callback

User callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if it is not needed

Definition at line 76 of file flexio\_i2s\_driver.h.

### 16.53.2.1.4 void\* callbackParam

Parameter for the callback function

Definition at line 80 of file flexio\_i2s\_driver.h.

### 16.53.2.1.5 flexio\_driver\_type\_t driverType

Driver type: interrupts/polling/DMA

Definition at line 69 of file flexio\_i2s\_driver.h.

### 16.53.2.1.6 uint8\_t rxDMAChannel

Rx DMA channel number. Only used in DMA mode

Definition at line 81 of file flexio\_i2s\_driver.h.

### 16.53.2.1.7 uint8\_t rxPin

Flexio pin to use for receive

Definition at line 73 of file flexio\_i2s\_driver.h.

### 16.53.2.1.8 uint8\_t sckPin

Flexio pin to use for serial clock

Definition at line 74 of file flexio\_i2s\_driver.h.

### 16.53.2.1.9 uint8\_t txDMAChannel

Tx DMA channel number. Only used in DMA mode

Definition at line 82 of file flexio\_i2s\_driver.h.

### 16.53.2.1.10 uint8\_t txPin

Flexio pin to use for transmit

Definition at line 72 of file flexio\_i2s\_driver.h.

### 16.53.2.1.11 uint8\_t wsPin

Flexio pin to use for word select

Definition at line 75 of file flexio\_i2s\_driver.h.

#### 16.53.2.2 struct flexio\_i2s\_slave\_user\_config\_t

Slave configuration structure.

This structure is used to provide configuration parameters for the flexio\_i2s slave at initialization time. Implements : flexio\_i2s\_slave\_user\_config\_t\_Class

Definition at line 92 of file flexio\_i2s\_driver.h.

##### Data Fields

- flexio\_driver\_type\_t driverType
- uint8\_t bitsWidth
- uint8\_t txPin
- uint8\_t rxPin
- uint8\_t sckPin
- uint8\_t wsPin
- i2s\_callback\_t callback
- void \* callbackParam
- uint8\_t rxDMAChannel
- uint8\_t txDMAChannel

##### Field Documentation

#### 16.53.2.2.1 uint8\_t bitsWidth

Number of bits in a word - multiple of 8

Definition at line 95 of file flexio\_i2s\_driver.h.

#### 16.53.2.2.2 i2s\_callback\_t callback

User callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if it is not needed

Definition at line 100 of file flexio\_i2s\_driver.h.

#### 16.53.2.2.3 void\* callbackParam

Parameter for the callback function

Definition at line 104 of file flexio\_i2s\_driver.h.

#### 16.53.2.2.4 flexio\_driver\_type\_t driverType

Driver type: interrupts/polling/DMA

Definition at line 94 of file flexio\_i2s\_driver.h.

#### 16.53.2.2.5 uint8\_t rxDMAChannel

Rx DMA channel number. Only used in DMA mode

Definition at line 105 of file flexio\_i2s\_driver.h.

#### 16.53.2.2.6 uint8\_t rxPin

Flexio pin to use for receive

Definition at line 97 of file flexio\_i2s\_driver.h.

**16.53.2.2.7 uint8\_t sckPin**

Flexio pin to use for serial clock

Definition at line 98 of file flexio\_i2s\_driver.h.

**16.53.2.2.8 uint8\_t txDMAChannel**

Tx DMA channel number. Only used in DMA mode

Definition at line 106 of file flexio\_i2s\_driver.h.

**16.53.2.2.9 uint8\_t txPin**

Flexio pin to use for transmit

Definition at line 96 of file flexio\_i2s\_driver.h.

**16.53.2.2.10 uint8\_t wsPin**

Flexio pin to use for word select

Definition at line 99 of file flexio\_i2s\_driver.h.

**16.53.2.3 struct flexio\_i2s\_master\_state\_t**

Master internal context structure.

This structure is used by the driver for its internal logic. It must be provided by the application through the [FLEXIO\\_I2S\\_DRV\\_MasterInit\(\)](#) function, then it cannot be freed until the driver is de-initialized using [FLEXIO\\_I2S\\_DRV\\_MasterDeinit\(\)](#). The application should make no assumptions about the content of this structure.

Definition at line 118 of file flexio\_i2s\_driver.h.

**16.53.3 Typedef Documentation****16.53.3.1 typedef flexio\_i2s\_master\_state\_t flexio\_i2s\_slave\_state\_t**

Slave internal context structure.

This structure is used by the driver for its internal logic. It must be provided by the application through the [FLEXIO\\_I2S\\_DRV\\_SlaveInit\(\)](#) function, then it cannot be freed until the driver is de-initialized using [FLEXIO\\_I2S\\_DRV\\_SlaveDeinit\(\)](#). The application should make no assumptions about the content of this structure.

Definition at line 150 of file flexio\_i2s\_driver.h.

**16.53.4 Function Documentation****16.53.4.1 status\_t FLEXIO\_I2S\_DRV\_MasterDeinit ( flexio\_i2s\_master\_state\_t \* master )**

De-initialize the FLEXIO\_I2S master mode driver.

This function de-initializes the FLEXIO\_I2S driver in master mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

**Parameters**

<i>master</i>	Pointer to the FLEXIO_I2S master driver context structure.
---------------	--

**Returns**

Error or success status returned by API

Definition at line 1082 of file flexio\_i2s\_driver.c.

**16.53.4.2** `status_t FLEXIO_I2S_DRV_MasterGetBaudRate ( flexio_i2s_master_state_t * master, uint32_t * baudRate )`

Get the currently configured baud rate.

This function returns the currently configured I2S baud rate.

#### Parameters

<i>master</i>	Pointer to the FLEXIO_I2S master driver context structure.
<i>baudRate</i>	the current baud rate in hertz

#### Returns

Error or success status returned by API

Definition at line 1161 of file flexio\_i2s\_driver.c.

**16.53.4.3** `void FLEXIO_I2S_DRV_MasterGetDefaultConfig ( flexio_i2s_master_user_config_t * userConfigPtr )`

Returns default configuration structure for FLEXIO\_I2S master.

#### Parameters

<i>userConfigPtr</i>	Pointer to the FLEXIO_I2S user configuration structure.
----------------------	---

Definition at line 1655 of file flexio\_i2s\_driver.c.

**16.53.4.4** `status_t FLEXIO_I2S_DRV_MasterGetStatus ( flexio_i2s_master_state_t * master, uint32_t * bytesRemaining )`

Get the status of the current non-blocking I2S master transaction.

This function returns the current status of a non-blocking I2S master transaction. A return code of STATUS\_BUSY means the transfer is still in progress. Otherwise the function returns a status reflecting the outcome of the last transfer. When the driver is initialized in polling mode this function also advances the transfer by checking and handling the transmit and receive events, so it must be called frequently to avoid overflows or underflows.

#### Parameters

<i>master</i>	Pointer to the FLEXIO_I2S master driver context structure.
<i>bytesRemaining</i>	the remaining number of bytes to be transferred

#### Returns

Error or success status returned by API

Definition at line 1429 of file flexio\_i2s\_driver.c.

**16.53.4.5** `status_t FLEXIO_I2S_DRV_MasterInit ( uint32_t instance, const flexio_i2s_master_user_config_t * userConfigPtr, flexio_i2s_master_state_t * master )`

Initialize the FLEXIO\_I2S master mode driver.

This function initializes the FLEXIO\_I2S driver in master mode.

#### Parameters

<i>instance</i>	FLEXIO peripheral instance number
<i>userConfigPtr</i>	Pointer to the FLEXIO_I2S master user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.



<i>master</i>	Pointer to the FLEXIO_I2S master driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using <a href="#">FLEXIO_I2S_DRV_MasterDeinit()</a> .
---------------	--

**Returns**

Error or success status returned by API

Definition at line 991 of file flexio\_i2s\_driver.c.

**16.53.4.6** `status_t FLEXIO_I2S_DRV_MasterReceiveData ( flexio_i2s_master_state_t * master, uint8_t * rxBuff, uint32_t rxSize )`

Perform a non-blocking receive transaction on the I2S bus.

This function starts the reception of a block of data and returns immediately. The rest of the reception is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the FLEXIO\_I2S\_DRV\_MasterGetStatus() function (if the driver is initialized in polling mode). Use [FLEXIO\\_I2S\\_DRV\\_MasterGetStatus\(\)](#) to check the progress of the reception.

**Parameters**

<i>master</i>	Pointer to the FLEXIO_I2S master driver context structure.
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API

Definition at line 1302 of file flexio\_i2s\_driver.c.

**16.53.4.7** `status_t FLEXIO_I2S_DRV_MasterReceiveDataBlocking ( flexio_i2s_master_state_t * master, uint8_t * rxBuff, uint32_t rxSize, uint32_t timeout )`

Perform a blocking receive transaction on the I2S bus.

This function receives a block of data and only returns when the reception is complete.

**Parameters**

<i>master</i>	Pointer to the FLEXIO_I2S master driver context structure.
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred
<i>timeout</i>	timeout for the transfer in milliseconds

**Returns**

Error or success status returned by API

Definition at line 1372 of file flexio\_i2s\_driver.c.

**16.53.4.8** `status_t FLEXIO_I2S_DRV_MasterSendData ( flexio_i2s_master_state_t * master, const uint8_t * txBuff, uint32_t txSize )`

Perform a non-blocking send transaction on the I2S bus.

This function starts the transmission of a block of data and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the FLEXIO\_I2S\_DRV\_MasterGetStatus() function (if the driver is initialized in polling mode). Use [FLEXIO\\_I2S\\_DRV\\_MasterGetStatus\(\)](#) to check the progress of the transmission.

**Parameters**

<i>master</i>	Pointer to the FLEXIO_I2S master driver context structure.
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API

Definition at line 1201 of file flexio\_i2s\_driver.c.

**16.53.4.9** `status_t FLEXIO_I2S_DRV_MasterSendDataBlocking ( flexio_i2s_master_state_t * master, const uint8_t * txBuff, uint32_t txSize, uint32_t timeout )`

Perform a blocking send transaction on the I2S bus.

This function sends a block of data, and only returns when the transmission is complete.

**Parameters**

<i>master</i>	Pointer to the FLEXIO_I2S master driver context structure.
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred
<i>timeout</i>	timeout for the transfer in milliseconds

**Returns**

Error or success status returned by API

Definition at line 1269 of file flexio\_i2s\_driver.c.

**16.53.4.10** `status_t FLEXIO_I2S_DRV_MasterSetConfig ( flexio_i2s_master_state_t * master, uint32_t baudRate, uint8_t bitsWidth )`

Set the baud rate and bit width for any subsequent I2S communication.

This function sets the baud rate (SCK frequency) and bit width for the I2S master. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call [FLEXIO\\_I2S\\_DRV\\_MasterGetBaudRate\(\)](#) after [FLEXIO\\_I2S\\_DRV\\_MasterSetConfig\(\)](#) to check what baud rate was actually set.

**Parameters**

<i>master</i>	Pointer to the FLEXIO_I2S master driver context structure.
<i>baudRate</i>	the desired baud rate in hertz
<i>bitsWidth</i>	number of bits per word

**Returns**

Error or success status returned by API

Definition at line 1108 of file flexio\_i2s\_driver.c.

**16.53.4.11** `status_t FLEXIO_I2S_DRV_MasterSetRxBuffer ( flexio_i2s_master_state_t * master, uint8_t * rxBuff, uint32_t rxSize )`

Provide a buffer for receiving data.

This function can be used to provide a new buffer for receiving data to the driver. It can be called from the user callback when event STATUS\_I2S\_RX\_OVERRUN is reported. This way the reception will continue without interruption.

## Parameters

<i>master</i>	Pointer to the FLEXIO_I2S master driver context structure.
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred

## Returns

Error or success status returned by API

Definition at line 1484 of file flexio\_i2s\_driver.c.

**16.53.4.12** `status_t FLEXIO_I2S_DRV_MasterSetTxBuffer ( flexio_i2s_master_state_t * master, const uint8_t * txBuff, uint32_t txSize )`

Provide a buffer for transmitting data.

This function can be used to provide a new buffer for transmitting data to the driver. It can be called from the user callback when event STATUS\_I2S\_TX\_UNDERRUN is reported. This way the transmission will continue without interruption.

## Parameters

<i>master</i>	Pointer to the FLEXIO_I2S master driver context structure.
<i>txBuff</i>	pointer to the buffer containing transmit data
<i>txSize</i>	length in bytes of the data to be transferred

## Returns

Error or success status returned by API

Definition at line 1506 of file flexio\_i2s\_driver.c.

**16.53.4.13** `status_t FLEXIO_I2S_DRV_MasterTransferAbort ( flexio_i2s_master_state_t * master )`

Aborts a non-blocking I2S master transaction.

This function aborts a non-blocking I2S transfer.

## Parameters

<i>master</i>	Pointer to the FLEXIO_I2S master driver context structure.
---------------	--

## Returns

Error or success status returned by API

Definition at line 1405 of file flexio\_i2s\_driver.c.

**16.53.4.14** `status_t FLEXIO_I2S_DRV_SlaveDeinit ( flexio_i2s_slave_state_t * slave )`

De-initialize the FLEXIO\_I2S slave mode driver.

This function de-initializes the FLEXIO\_I2S driver in slave mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

## Parameters

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
--------------	---

## Returns

Error or success status returned by API

This function de-initializes the FLEXIO\_I2S driver in slave mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
--------------	---

**Returns**

Error or success status returned by API Implements : FLEXIO\_I2S\_DRV\_SlaveDeinit\_Activity

Definition at line 1708 of file flexio\_i2s\_driver.c.

**16.53.4.15** void FLEXIO\_I2S\_DRV\_SlaveGetDefaultConfig ( flexio\_i2s\_slave\_user\_config\_t \* userConfigPtr )

Returns default configuration structure for FLEXIO\_I2S slave.

**Parameters**

<i>userConfigPtr</i>	Pointer to the FLEXIO_I2S user configuration structure.
----------------------	---

Definition at line 1680 of file flexio\_i2s\_driver.c.

**16.53.4.16** status\_t FLEXIO\_I2S\_DRV\_SlaveGetStatus ( flexio\_i2s\_slave\_state\_t \* slave, uint32\_t \* bytesRemaining )

Get the status of the current non-blocking I2S slave transaction.

This function returns the current status of a non-blocking I2S slave transaction. A return code of STATUS\_BUSY means the transfer is still in progress. Otherwise the function returns a status reflecting the outcome of the last transfer. When the driver is initialized in polling mode this function also advances the transfer by checking and handling the transmit and receive events, so it must be called frequently to avoid overflows or underflows.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>bytesRemaining</i>	the remaining number of bytes to be transferred

**Returns**

Error or success status returned by API

This function returns the current status of a non-blocking I2S slave transaction. A return code of STATUS\_BUSY means the transfer is still in progress. Otherwise the function returns a status reflecting the outcome of the last transfer. When the driver is initialized in polling mode this function also advances the transfer by checking and handling the transmit and receive events, so it must be called frequently to avoid overflows or underflows.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>bytesRemaining</i>	the remaining number of bytes to be transferred

**Returns**

Error or success status returned by API Implements : FLEXIO\_I2S\_DRV\_SlaveGetStatus\_Activity

Definition at line 1834 of file flexio\_i2s\_driver.c.

**16.53.4.17** status\_t FLEXIO\_I2S\_DRV\_Slavelnit ( uint32\_t instance, const flexio\_i2s\_slave\_user\_config\_t \* userConfigPtr, flexio\_i2s\_slave\_state\_t \* slave )

Initialize the FLEXIO\_I2S slave mode driver.

This function initializes the FLEXIO\_I2S driver in slave mode.

## Parameters

<i>instance</i>	FLEXIO peripheral instance number
<i>userConfigPtr</i>	Pointer to the FLEXIO_I2S slave user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using <a href="#">FLEXIO_I2S_DRV_SlaveDeinit()</a> .

## Returns

Error or success status returned by API

Definition at line 1530 of file flexio\_i2s\_driver.c.

**16.53.4.18** `status_t FLEXIO_I2S_DRV_SlaveReceiveData ( flexio_i2s_slave_state_t * slave, uint8_t * rxBuff, uint32_t rxSize )`

Perform a non-blocking receive transaction on the I2S bus.

This function starts the reception of a block of data and returns immediately. The rest of the reception is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the [FLEXIO\\_I2S\\_DRV\\_SlaveGetStatus\(\)](#) function (if the driver is initialized in polling mode). Use [FLEXIO\\_I2S\\_DRV\\_SlaveGetStatus\(\)](#) to check the progress of the reception.

## Parameters

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred

## Returns

Error or success status returned by API

This function starts the reception of a block of data and returns immediately. The rest of the reception is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the [FLEXIO\\_I2S\\_DRV\\_SlaveGetStatus\(\)](#) function (if the driver is initialized in polling mode). Use [FLEXIO\\_I2S\\_DRV\\_SlaveGetStatus\(\)](#) to check the progress of the reception.

## Parameters

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred

## Returns

Error or success status returned by API Implements : [FLEXIO\\_I2S\\_DRV\\_SlaveReceiveData\\_Activity](#)

Definition at line 1775 of file flexio\_i2s\_driver.c.

**16.53.4.19** `status_t FLEXIO_I2S_DRV_SlaveReceiveDataBlocking ( flexio_i2s_slave_state_t * slave, uint8_t * rxBuff, uint32_t rxSize, uint32_t timeout )`

Perform a blocking receive transaction on the I2S bus.

This function receives a block of data and only returns when the reception is complete.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred
<i>timeout</i>	timeout for the transfer in milliseconds

**Returns**

Error or success status returned by API

This function receives a block of data and only returns when the reception is complete.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred
<i>timeout</i>	timeout for the transfer in milliseconds

**Returns**

Error or success status returned by API Implements : FLEXIO\_I2S\_DRV\_SlaveReceiveDataBlocking\_Activity

Definition at line 1795 of file flexio\_i2s\_driver.c.

**16.53.4.20** `status_t FLEXIO_I2S_DRV_SlaveSendData ( flexio_i2s_slave_state_t * slave, const uint8_t * txBuff, uint32_t txSize )`

Perform a non-blocking send transaction on the I2S bus.

This function starts the transmission of a block of data and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the FLEXIO\_I2S\_DRV\_SlaveGetStatus function (if the driver is initialized in polling mode). Use [FLEXIO\\_I2S\\_DRV\\_SlaveGetStatus\(\)](#) to check the progress of the transmission.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API

This function starts the transmission of a block of data and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the FLEXIO\_I2S\_DRV\_SlaveGetStatus function (if the driver is initialized in polling mode). Use [FLEXIO\\_I2S\\_DRV\\_SlaveGetStatus\(\)](#) to check the progress of the transmission.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API Implements : FLEXIO\_I2S\_DRV\_SlaveSendData\_Activity

Definition at line 1729 of file flexio\_i2s\_driver.c.

**16.53.4.21** `status_t FLEXIO_I2S_DRV_SlaveSendDataBlocking ( flexio_i2s_slave_state_t * slave, const uint8_t * txBuff, uint32_t txSize, uint32_t timeout )`

Perform a blocking send transaction on the I2S bus.

This function sends a block of data, and only returns when the transmission is complete.

#### Parameters

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred
<i>timeout</i>	timeout for the transfer in milliseconds

#### Returns

Error or success status returned by API

This function sends a block of data, and only returns when the transmission is complete.

#### Parameters

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred
<i>timeout</i>	timeout for the transfer in milliseconds

#### Returns

Error or success status returned by API Implements : FLEXIO\_I2S\_DRV\_SlaveSendDataBlocking\_Activity

Definition at line 1751 of file flexio\_i2s\_driver.c.

**16.53.4.22** `status_t FLEXIO_I2S_DRV_SlaveSetConfig ( flexio_i2s_slave_state_t * slave, uint8_t bitsWidth )`

Set the bit width for any subsequent I2S communication.

This function sets the bit width for the I2S slave.

#### Parameters

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>bitsWidth</i>	number of bits per word

#### Returns

Error or success status returned by API

Definition at line 1612 of file flexio\_i2s\_driver.c.

**16.53.4.23** `status_t FLEXIO_I2S_DRV_SlaveSetRxBuffer ( flexio_i2s_slave_state_t * slave, uint8_t * rxBuff, uint32_t rxSize )`

Provide a buffer for receiving data.

This function can be used to provide a driver with a new buffer for receiving data. It can be called from the user callback when event STATUS\_I2S\_RX\_OVERRUN is reported. This way the reception will continue without interruption.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API

This function can be used to provide a driver with a new buffer for receiving data. It can be called from the user callback when event STATUS\_I2S\_RX\_OVERRUN is reported. This way the reception will continue without interruption.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API Implements : FLEXIO\_I2S\_DRV\_SlaveSetRxBuffer\_Activity

Definition at line 1853 of file flexio\_i2s\_driver.c.

**16.53.4.24** `status_t FLEXIO_I2S_DRV_SlaveSetTxBuffer ( flexio_i2s_slave_state_t * slave, const uint8_t * txBuff, uint32_t txSize )`

Provide a buffer for transmitting data.

This function can be used to provide a driver with a new buffer for transmitting data. It can be called from the user callback when event STATUS\_I2S\_TX\_UNDERRUN is reported. This way the transmission will continue without interruption.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>txBuff</i>	pointer to the buffer containing transmit data
<i>txSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API

This function can be used to provide a driver with a new buffer for transmitting data. It can be called from the user callback when event STATUS\_I2S\_TX\_UNDERRUN is reported. This way the transmission will continue without interruption.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
<i>txBuff</i>	pointer to the buffer containing transmit data
<i>txSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API Implements : FLEXIO\_I2S\_DRV\_SlaveSetTxBuffer\_Activity

Definition at line 1874 of file flexio\_i2s\_driver.c.



16.53.4.25 `status_t FLEXIO_I2S_DRV_SlaveTransferAbort ( flexio_i2s_slave_state_t * slave )`

Aborts a non-blocking I2S slave transaction.

This function aborts a non-blocking I2S transfer.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
--------------	---

**Returns**

Error or success status returned by API

This function aborts a non-blocking I2S transfer.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_I2S slave driver context structure.
--------------	---

**Returns**

Error or success status returned by API Implements : FLEXIO\_I2S\_DRV\_SlaveTransferAbort\_Activity

Definition at line 1813 of file flexio\_i2s\_driver.c.

## 16.54 FlexIO SPI Driver

### 16.54.1 Detailed Description

SPI communication over FlexIO module (FLEXIO\_SPI)

The FLEXIO\_SPI Driver allows communication on an SPI bus using the FlexIO module in the S32K1xx processors.

#### Features

- Master or slave operation
- Interrupt, DMA or polling mode
- Provides blocking and non-blocking transfer functions
- Configurable baud rate
- Configurable clock polarity and phase
- Configurable bit order and data size

#### Functionality

Before using any Flexio driver the device must first be initialized using function `FLEXIO_DRV_InitDevice`. Then the FLEXIO\_SPI Driver must be initialized, using functions `FLEXIO_SPI_DRV_MasterInit()` or `FLEXIO_SPI_DRV_SlaveInit()`. It is possible to use more driver instances on the same FlexIO device, as long as sufficient resources are available. Different driver instances on the same FlexIO device can function independently of each other. When it is no longer needed, the driver can be de-initialized, using `FLEXIO_SPI_DRV_MasterDeinit()` or `FLEXIO_SPI_DRV_SlaveDeinit()`. This will release the hardware resources, allowing other driver instances to be initialized other.

#### Master Mode

Master Mode provides functions for transmitting or receiving data to/from an SPI slave. Baud rate is provided at initialization time through the master configuration structure, but can be changed at runtime by using `FLEXIO_SPI_DRV_MasterSetBaudRate()` function. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call `FLEXIO_SPI_DRV_MasterGetBaudRate()` after `FLEXIO_SPI_DRV_MasterSetBaudRate()` to check what baud rate was actually set.

To send or receive data, use function `FLEXIO_SPI_DRV_MasterTransfer()`. The transmit and receive buffers, together with parameters for the transfer are provided through the `flexio_spi_transfer_t` structure. If only transmit or receive is desired, any one of the Rx/Tx buffers can be set to NULL. This driver does not support continuous send/receive using a user callback function. The callback function is only used to signal the end of a transfer.

Blocking operations will return only when the transfer is completed, either successfully or with error. Non-blocking operations will initiate the transfer and return `STATUS_SUCCESS`, but the module is still busy with the transfer and another transfer can't be initiated until the current transfer is complete. The application will be notified through the user callback when the transfer completes, or it can check the status of the current transfer by calling `FLEXIO_SPI_DRV_MasterGetStatus()`. If the transfer is still ongoing this function will return `STATUS_BUSY`. If the transfer is completed, the function will return either `STATUS_SUCCESS` or an error code, depending on the outcome of the last transfer.

The driver supports interrupt, DMA and polling mode. In polling mode the function `FLEXIO_SPI_DRV_MasterGetStatus()` ensures the progress of the transfer by checking and handling transmit and receive events reported by the FlexIO module. The application should ensure that this function is called often enough (at least once per transferred byte) to avoid Tx underflows or Rx overflows. In DMA mode the DMA channels that will be used by the driver are received through the configuration structure. The channels must be initialized by the application before the `flexio_spi` driver is initialized. The `flexio_spi` driver will only set the DMA request source.

## Slave Mode

Slave Mode is very similar to master mode, the main difference being that the `FLEXIO_SPI_DRV_SlaveInit()` function initializes the FlexIO module to use the clock signal received from the master instead of generating it. Consequently, there is no `SetBaudRate` function in slave mode. Other than that, the slave mode offers a similar interface to the master mode. `FLEXIO_SPI_DRV_MasterTransfer()` can be used to initiate transfers, and `FLEXIO_SPI_DRV_SlaveGetStatus()` is used to check the status of the transfer and advance the transfer in polling mode. All other specifications from the Master Mode description apply for Slave Mode too

## Important Notes

- Before using the FLEXIO\_SPI Driver the protocol clock of the module must be configured. Refer to Clock Manager for clock configuration.
- Before using the FLEXIO\_SPI Driver the pins must be routed to the FlexIO module. Refer to PINS Driver for pin routing configuration. Note that any of the available FlexIO pins can be used for MOSI, MISO, SCK and SS (configurable at initialization time).
- The driver enables the interrupts for the corresponding FlexIO module, but any interrupt priority setting must be done by the application.
- The driver does not support back-to-back transmission mode for CPHA = 1
- The driver does not support configurable polarity for SS signal (only active-low is supported)
- Timeout feature for blocking transfers does not work in polling mode.
- This driver needs two shifters and two timers for its operation. Initialization will fail if there are not enough shifters and timers available on the FlexIO device.
- This driver needs two DMA channels for its operation when it is initialized in DMA mode. The DMA channels must be initialized by the application before initializing the driver. Refer to EDMA driver for DMA channels initialization.
- If the application uses an RTOS, this driver uses a semaphore for blocking transfers. Initialization will fail if the semaphore cannot be created. If the driver uses polling mode no semaphore is used.
- If the application uses an RTOS, the FlexIO drivers use a mutex for channel allocation. Only one mutex per device is needed, not per driver instance. Device initialization will fail if the mutex cannot be created.
- For transfers where the data size is more than 1 byte (transferSize is 2 or 4) the driver assumes that the data buffers are defined with the proper type (`uint16_t` or `uint32_t`) and are properly aligned.

## Integration guideline

### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\drivers\src\flexio\flexio_common.c
${S32SDK_PATH}\platform\drivers\src\flexio\flexio_spi_driver.c
```

### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc
${S32SDK_PATH}\platform\drivers\src\flexio
```

### Compile symbols

No special symbols are required for this component

## Dependencies

[Clock Manager OS Interface \(OSIF\)](#) [Interrupt Manager \(Interrupt\)](#) [Enhanced Direct Memory Access \(eDMA\)](#)

## Data Structures

- struct [flexio\\_spi\\_master\\_user\\_config\\_t](#)  
*Master configuration structure. [More...](#)*
- struct [flexio\\_spi\\_slave\\_user\\_config\\_t](#)  
*Slave configuration structure. [More...](#)*
- struct [flexio\\_spi\\_master\\_state\\_t](#)  
*Master internal context structure. [More...](#)*

## Typedefs

- typedef [flexio\\_spi\\_master\\_state\\_t](#) [flexio\\_spi\\_slave\\_state\\_t](#)  
*Slave internal context structure.*

## Enumerations

- enum [flexio\\_spi\\_transfer\\_bit\\_order\\_t](#) { [FLEXIO\\_SPI\\_TRANSFER\\_MSB\\_FIRST](#) = 0U, [FLEXIO\\_SPI\\_TRANSFER\\_LSB\\_FIRST](#) = 1U }  
*Order in which the data bits are transferred Implements : [flexio\\_spi\\_transfer\\_bit\\_order\\_t](#) Class.*
- enum [flexio\\_spi\\_transfer\\_size\\_t](#) { [FLEXIO\\_SPI\\_TRANSFER\\_1BYTE](#) = 1U, [FLEXIO\\_SPI\\_TRANSFER\\_2BYTE](#) = 2U, [FLEXIO\\_SPI\\_TRANSFER\\_4BYTE](#) = 4U }  
*Size of transferred data in bytes Implements : [flexio\\_spi\\_transfer\\_size\\_t](#) Class.*

## FLEXIO\_SPI Driver

- status\_t [FLEXIO\\_SPI\\_DRV\\_MasterInit](#) (uint32\_t instance, const [flexio\\_spi\\_master\\_user\\_config\\_t](#) \*userConfigPtr, [flexio\\_spi\\_master\\_state\\_t](#) \*master)  
*Initialize the FLEXIO\_SPI master mode driver.*
- status\_t [FLEXIO\\_SPI\\_DRV\\_MasterDeinit](#) ([flexio\\_spi\\_master\\_state\\_t](#) \*master)  
*De-initialize the FLEXIO\_SPI master mode driver.*
- status\_t [FLEXIO\\_SPI\\_DRV\\_MasterSetBaudRate](#) ([flexio\\_spi\\_master\\_state\\_t](#) \*master, uint32\_t baudRate)  
*Set the baud rate for any subsequent SPI communication.*
- status\_t [FLEXIO\\_SPI\\_DRV\\_MasterGetBaudRate](#) ([flexio\\_spi\\_master\\_state\\_t](#) \*master, uint32\_t \*baudRate)  
*Get the currently configured baud rate.*
- status\_t [FLEXIO\\_SPI\\_DRV\\_MasterTransfer](#) ([flexio\\_spi\\_master\\_state\\_t](#) \*master, const uint8\_t \*txData, uint8\_t \*rxData, uint32\_t dataSize)  
*Perform a non-blocking SPI master transaction.*
- status\_t [FLEXIO\\_SPI\\_DRV\\_MasterTransferBlocking](#) ([flexio\\_spi\\_master\\_state\\_t](#) \*master, const uint8\_t \*txData, uint8\_t \*rxData, uint32\_t dataSize, uint32\_t timeout)  
*Perform a blocking SPI master transaction.*
- status\_t [FLEXIO\\_SPI\\_DRV\\_MasterTransferAbort](#) ([flexio\\_spi\\_master\\_state\\_t](#) \*master)  
*Aborts a non-blocking SPI master transaction.*
- status\_t [FLEXIO\\_SPI\\_DRV\\_MasterGetStatus](#) ([flexio\\_spi\\_master\\_state\\_t](#) \*master, uint32\_t \*bytesRemaining)  
*Get the status of the current non-blocking SPI master transaction.*
- status\_t [FLEXIO\\_SPI\\_DRV\\_SlaveInit](#) (uint32\_t instance, const [flexio\\_spi\\_slave\\_user\\_config\\_t](#) \*userConfigPtr, [flexio\\_spi\\_slave\\_state\\_t](#) \*slave)

- Initialize the FLEXIO\_SPI slave mode driver.*
- status\_t [FLEXIO\\_SPI\\_DRV\\_SlaveDeinit](#) ([flexio\\_spi\\_slave\\_state\\_t](#) \*slave)
- De-initialize the FLEXIO\_SPI slave mode driver.*
- status\_t [FLEXIO\\_SPI\\_DRV\\_SlaveTransfer](#) ([flexio\\_spi\\_slave\\_state\\_t](#) \*slave, const uint8\_t \*txData, uint8\_t \*rxData, uint32\_t dataSize)
- Perform a non-blocking SPI slave transaction.*
- status\_t [FLEXIO\\_SPI\\_DRV\\_SlaveTransferBlocking](#) ([flexio\\_spi\\_slave\\_state\\_t](#) \*slave, const uint8\_t \*txData, uint8\_t \*rxData, uint32\_t dataSize, uint32\_t timeout)
- Perform a blocking SPI slave transaction.*
- status\_t [FLEXIO\\_SPI\\_DRV\\_SlaveTransferAbort](#) ([flexio\\_spi\\_slave\\_state\\_t](#) \*slave)
- Aborts a non-blocking SPI slave transaction.*
- status\_t [FLEXIO\\_SPI\\_DRV\\_SlaveGetStatus](#) ([flexio\\_spi\\_slave\\_state\\_t](#) \*slave, uint32\_t \*bytesRemaining)
- Get the status of the current non-blocking SPI slave transaction.*
- void [FLEXIO\\_SPI\\_DRV\\_MasterGetDefaultConfig](#) ([flexio\\_spi\\_master\\_user\\_config\\_t](#) \*userConfigPtr)
- Returns default configuration structure for FLEXIO\_SPI master.*
- void [FLEXIO\\_SPI\\_DRV\\_SlaveGetDefaultConfig](#) ([flexio\\_spi\\_slave\\_user\\_config\\_t](#) \*userConfigPtr)
- Returns default configuration structure for FLEXIO\_SPI slave.*

## 16.54.2 Data Structure Documentation

### 16.54.2.1 struct flexio\_spi\_master\_user\_config\_t

Master configuration structure.

This structure is used to provide configuration parameters for the flexio\_spi master at initialization time. Implements : flexio\_spi\_master\_user\_config\_t\_Class

Definition at line 67 of file flexio\_spi\_driver.h.

#### Data Fields

- uint32\_t [baudRate](#)
- [flexio\\_driver\\_type\\_t](#) driverType
- [flexio\\_spi\\_transfer\\_bit\\_order\\_t](#) bitOrder
- [flexio\\_spi\\_transfer\\_size\\_t](#) transferSize
- uint8\_t [clockPolarity](#)
- uint8\_t [clockPhase](#)
- uint8\_t [mosiPin](#)
- uint8\_t [misoPin](#)
- uint8\_t [sckPin](#)
- uint8\_t [ssPin](#)
- spi\_callback\_t [callback](#)
- void \* [callbackParam](#)
- uint8\_t [rxDMACHannel](#)
- uint8\_t [txDMACHannel](#)

#### Field Documentation

##### 16.54.2.1.1 uint32\_t baudRate

Baud rate in hertz

Definition at line 69 of file flexio\_spi\_driver.h.

##### 16.54.2.1.2 flexio\_spi\_transfer\_bit\_order\_t bitOrder

Bit order: LSB-first / MSB-first

Definition at line 71 of file flexio\_spi\_driver.h.

**16.54.2.1.3 spi\_callback\_t callback**

User callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if it is not needed

Definition at line 79 of file flexio\_spi\_driver.h.

**16.54.2.1.4 void\* callbackParam**

Parameter for the callback function

Definition at line 83 of file flexio\_spi\_driver.h.

**16.54.2.1.5 uint8\_t clockPhase**

Clock Phase (CPHA) 0 = sample on leading clock edge; 1 = sample on trailing clock edge

Definition at line 74 of file flexio\_spi\_driver.h.

**16.54.2.1.6 uint8\_t clockPolarity**

Clock Polarity (CPOL) 0 = active-high clock; 1 = active-low clock

Definition at line 73 of file flexio\_spi\_driver.h.

**16.54.2.1.7 flexio\_driver\_type\_t driverType**

Driver type: interrupts/polling/DMA

Definition at line 70 of file flexio\_spi\_driver.h.

**16.54.2.1.8 uint8\_t misoPin**

Flexio pin to use as MISO pin

Definition at line 76 of file flexio\_spi\_driver.h.

**16.54.2.1.9 uint8\_t mosiPin**

Flexio pin to use as MOSI pin

Definition at line 75 of file flexio\_spi\_driver.h.

**16.54.2.1.10 uint8\_t rxDMAChannel**

Rx DMA channel number. Only used in DMA mode

Definition at line 84 of file flexio\_spi\_driver.h.

**16.54.2.1.11 uint8\_t sckPin**

Flexio pin to use as SCK pin

Definition at line 77 of file flexio\_spi\_driver.h.

**16.54.2.1.12 uint8\_t ssPin**

Flexio pin to use as SS pin

Definition at line 78 of file flexio\_spi\_driver.h.

**16.54.2.1.13 flexio\_spi\_transfer\_size\_t transferSize**

Transfer size in bytes: 1/2/4

Definition at line 72 of file flexio\_spi\_driver.h.

#### 16.54.2.1.14 uint8\_t txDMAChannel

Tx DMA channel number. Only used in DMA mode

Definition at line 85 of file flexio\_spi\_driver.h.

#### 16.54.2.2 struct flexio\_spi\_slave\_user\_config\_t

Slave configuration structure.

This structure is used to provide configuration parameters for the flexio\_spi slave at initialization time. Implements : flexio\_spi\_slave\_user\_config\_t\_Class

Definition at line 94 of file flexio\_spi\_driver.h.

#### Data Fields

- [flexio\\_driver\\_type\\_t driverType](#)
- [flexio\\_spi\\_transfer\\_bit\\_order\\_t bitOrder](#)
- [flexio\\_spi\\_transfer\\_size\\_t transferSize](#)
- [uint8\\_t clockPolarity](#)
- [uint8\\_t clockPhase](#)
- [uint8\\_t mosiPin](#)
- [uint8\\_t misoPin](#)
- [uint8\\_t sckPin](#)
- [uint8\\_t ssPin](#)
- [spi\\_callback\\_t callback](#)
- [void \\* callbackParam](#)
- [uint8\\_t rxDMAChannel](#)
- [uint8\\_t txDMAChannel](#)

#### Field Documentation

##### 16.54.2.2.1 flexio\_spi\_transfer\_bit\_order\_t bitOrder

Bit order: LSB-first / MSB-first

Definition at line 97 of file flexio\_spi\_driver.h.

##### 16.54.2.2.2 spi\_callback\_t callback

User callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if it is not needed

Definition at line 105 of file flexio\_spi\_driver.h.

##### 16.54.2.2.3 void\* callbackParam

Parameter for the callback function

Definition at line 109 of file flexio\_spi\_driver.h.

##### 16.54.2.2.4 uint8\_t clockPhase

Clock Phase (CPHA) 0 = sample on leading clock edge; 1 = sample on trailing clock edge

Definition at line 100 of file flexio\_spi\_driver.h.

##### 16.54.2.2.5 uint8\_t clockPolarity

Clock Polarity (CPOL) 0 = active-low clock; 1 = active-high clock

Definition at line 99 of file flexio\_spi\_driver.h.



**16.54.2.2.6 flexio\_driver\_type\_t driverType**

Driver type: interrupts/polling/DMA

Definition at line 96 of file flexio\_spi\_driver.h.

**16.54.2.2.7 uint8\_t misoPin**

Flexio pin to use as MISO pin

Definition at line 102 of file flexio\_spi\_driver.h.

**16.54.2.2.8 uint8\_t mosiPin**

Flexio pin to use as MOSI pin

Definition at line 101 of file flexio\_spi\_driver.h.

**16.54.2.2.9 uint8\_t rxDMAChannel**

Rx DMA channel number. Only used in DMA mode

Definition at line 110 of file flexio\_spi\_driver.h.

**16.54.2.2.10 uint8\_t sckPin**

Flexio pin to use as SCK pin

Definition at line 103 of file flexio\_spi\_driver.h.

**16.54.2.2.11 uint8\_t ssPin**

Flexio pin to use as SS pin

Definition at line 104 of file flexio\_spi\_driver.h.

**16.54.2.2.12 flexio\_spi\_transfer\_size\_t transferSize**

Transfer size in bytes: 1/2/4

Definition at line 98 of file flexio\_spi\_driver.h.

**16.54.2.2.13 uint8\_t txDMAChannel**

Tx DMA channel number. Only used in DMA mode

Definition at line 111 of file flexio\_spi\_driver.h.

**16.54.2.3 struct flexio\_spi\_master\_state\_t**

Master internal context structure.

This structure is used by the master-mode driver for its internal logic. It must be provided by the application through the [FLEXIO\\_SPI\\_DRV\\_MasterInit\(\)](#) function, then it cannot be freed until the driver is de-initialized using [FLEXIO\\_SPI\\_DRV\\_MasterDeinit\(\)](#). The application should make no assumptions about the content of this structure.

Definition at line 123 of file flexio\_spi\_driver.h.

**16.54.3 Typedef Documentation****16.54.3.1 typedef flexio\_spi\_master\_state\_t flexio\_spi\_slave\_state\_t**

Slave internal context structure.

This structure is used by the slave-mode driver for its internal logic. It must be provided by the application through the [FLEXIO\\_SPI\\_DRV\\_SlaveInit\(\)](#) function, then it cannot be freed until the driver is de-initialized using [FLEXIO\\_SPI\\_DRV\\_SlaveDeinit\(\)](#).

[\\_SPI\\_DRV\\_SlaveDeinit\(\)](#). The application should make no assumptions about the content of this structure.

Definition at line 155 of file flexio\_spi\_driver.h.

#### 16.54.4 Enumeration Type Documentation

##### 16.54.4.1 enum flexio\_spi\_transfer\_bit\_order\_t

Order in which the data bits are transferred Implements : flexio\_spi\_transfer\_bit\_order\_t\_Class.

###### Enumerator

**FLEXIO\_SPI\_TRANSFER\_MSB\_FIRST** Transmit data starting with most significant bit

**FLEXIO\_SPI\_TRANSFER\_LSB\_FIRST** Transmit data starting with least significant bit

Definition at line 39 of file flexio\_spi\_driver.h.

##### 16.54.4.2 enum flexio\_spi\_transfer\_size\_t

Size of transferred data in bytes Implements : flexio\_spi\_transfer\_size\_t\_Class.

###### Enumerator

**FLEXIO\_SPI\_TRANSFER\_1BYTE** Data size is 1-byte

**FLEXIO\_SPI\_TRANSFER\_2BYTE** Data size is 2-bytes

**FLEXIO\_SPI\_TRANSFER\_4BYTE** Data size is 4-bytes

Definition at line 48 of file flexio\_spi\_driver.h.

#### 16.54.5 Function Documentation

##### 16.54.5.1 status\_t FLEXIO\_SPI\_DRV\_MasterDeinit ( flexio\_spi\_master\_state\_t \* master )

De-initialize the FLEXIO\_SPI master mode driver.

This function de-initializes the FLEXIO\_SPI driver in master mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

###### Parameters

<i>master</i>	Pointer to the FLEXIO_SPI master driver context structure.
---------------	--

###### Returns

Error or success status returned by API

Definition at line 982 of file flexio\_spi\_driver.c.

##### 16.54.5.2 status\_t FLEXIO\_SPI\_DRV\_MasterGetBaudRate ( flexio\_spi\_master\_state\_t \* master, uint32\_t \* baudRate )

Get the currently configured baud rate.

This function returns the currently configured SPI baud rate.

###### Parameters

<i>master</i>	Pointer to the FLEXIO_SPI master driver context structure.
<i>baudRate</i>	the current baud rate in hertz

**Returns**

Error or success status returned by API

Definition at line 1054 of file flexio\_spi\_driver.c.

#### 16.54.5.3 void FLEXIO\_SPI\_DRV\_MasterGetDefaultConfig ( flexio\_spi\_master\_user\_config\_t \* userConfigPtr )

Returns default configuration structure for FLEXIO\_SPI master.

**Parameters**

<i>userConfigPtr</i>	Pointer to the FLEXIO_SPI user configuration structure.
----------------------	---

Definition at line 1352 of file flexio\_spi\_driver.c.

#### 16.54.5.4 status\_t FLEXIO\_SPI\_DRV\_MasterGetStatus ( flexio\_spi\_master\_state\_t \* master, uint32\_t \* bytesRemaining )

Get the status of the current non-blocking SPI master transaction.

This function returns the current status of a non-blocking SPI master transaction. A return code of STATUS\_BUSY means the transfer is still in progress. Otherwise the function returns a status reflecting the outcome of the last transfer. When the driver is initialized in polling mode this function also advances the transfer by checking and handling the transmit and receive events, so it must be called frequently to avoid overflows or underflows.

**Parameters**

<i>master</i>	Pointer to the FLEXIO_SPI master driver context structure.
<i>bytesRemaining</i>	the remaining number of bytes to be transferred

**Returns**

Error or success status returned by API

Definition at line 1225 of file flexio\_spi\_driver.c.

#### 16.54.5.5 status\_t FLEXIO\_SPI\_DRV\_MasterInit ( uint32\_t instance, const flexio\_spi\_master\_user\_config\_t \* userConfigPtr, flexio\_spi\_master\_state\_t \* master )

Initialize the FLEXIO\_SPI master mode driver.

This function initializes the FLEXIO\_SPI driver in master mode.

**Parameters**

<i>instance</i>	FLEXIO peripheral instance number
<i>userConfigPtr</i>	Pointer to the FLEXIO_SPI master user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
<i>master</i>	Pointer to the FLEXIO_SPI master driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using <a href="#">FLEXIO_SPI_DRV_MasterDeinit()</a> .

**Returns**

Error or success status returned by API

Definition at line 893 of file flexio\_spi\_driver.c.

#### 16.54.5.6 `status_t FLEXIO_SPI_DRV_MasterSetBaudRate ( flexio_spi_master_state_t * master, uint32_t baudRate )`

Set the baud rate for any subsequent SPI communication.

This function sets the baud rate for the SPI master. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call [FLEXIO\\_SPI\\_DRV\\_MasterGetBaudRate\(\)](#) after [FLEXIO\\_SPI\\_DRV\\_MasterSetBaudRate\(\)](#) to check what baud rate was actually set.

##### Parameters

<i>master</i>	Pointer to the FLEXIO_SPI master driver context structure.
<i>baudRate</i>	the desired baud rate in hertz

##### Returns

Error or success status returned by API

Definition at line 1009 of file flexio\_spi\_driver.c.

#### 16.54.5.7 `status_t FLEXIO_SPI_DRV_MasterTransfer ( flexio_spi_master_state_t * master, const uint8_t * txData, uint8_t * rxData, uint32_t dataSize )`

Perform a non-blocking SPI master transaction.

This function performs an SPI full-duplex transaction, transmit and receive in parallel. If only transmit or receive are required, it is possible to provide NULL pointers for txData or rxData. The transfer is non-blocking, the function only initiates the transfer and then returns, leaving the transfer to complete asynchronously). [FLEXIO\\_SPI\\_DRV\\_MasterGetStatus\(\)](#) can be called to check the status of the transfer.

##### Parameters

<i>master</i>	Pointer to the FLEXIO_SPI master driver context structure.
<i>txData</i>	pointer to the data to be transmitted
<i>rxData</i>	pointer to the buffer where to store received data
<i>dataSize</i>	length in bytes of the data to be transferred

##### Returns

Error or success status returned by API

Definition at line 1095 of file flexio\_spi\_driver.c.

#### 16.54.5.8 `status_t FLEXIO_SPI_DRV_MasterTransferAbort ( flexio_spi_master_state_t * master )`

Aborts a non-blocking SPI master transaction.

This function aborts a non-blocking SPI transfer.

##### Parameters

<i>master</i>	Pointer to the FLEXIO_SPI master driver context structure.
---------------	--

##### Returns

Error or success status returned by API

Definition at line 1201 of file flexio\_spi\_driver.c.

#### 16.54.5.9 `status_t FLEXIO_SPI_DRV_MasterTransferBlocking ( flexio_spi_master_state_t * master, const uint8_t * txData, uint8_t * rxData, uint32_t dataSize, uint32_t timeout )`

Perform a blocking SPI master transaction.

This function performs an SPI full-duplex transaction, transmit and receive in parallel. If only transmit or receive are required, it is possible to provide NULL pointers for txData or rxData. The transfer is blocking, the function only returns when the transfer is complete.

#### Parameters

<i>master</i>	Pointer to the FLEXIO_SPI master driver context structure.
<i>txData</i>	pointer to the data to be transmitted
<i>rxData</i>	pointer to the buffer where to store received data
<i>dataSize</i>	length in bytes of the data to be transferred
<i>timeout</i>	timeout for the transfer in milliseconds

#### Returns

Error or success status returned by API

Definition at line 1160 of file flexio\_spi\_driver.c.

**16.54.5.10** `status_t FLEXIO_SPI_DRV_SlaveDeinit ( flexio_spi_slave_state_t * slave )`

De-initialize the FLEXIO\_SPI slave mode driver.

This function de-initializes the FLEXIO\_SPI driver in slave mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

#### Parameters

<i>slave</i>	Pointer to the FLEXIO_SPI slave driver context structure.
--------------	---

#### Returns

Error or success status returned by API

This function de-initializes the FLEXIO\_SPI driver in slave mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

#### Parameters

<i>slave</i>	Pointer to the FLEXIO_SPI slave driver context structure.
--------------	---

#### Returns

Error or success status returned by API Implements : FLEXIO\_SPI\_DRV\_SlaveDeinit\_Activity

Definition at line 1411 of file flexio\_spi\_driver.c.

**16.54.5.11** `void FLEXIO_SPI_DRV_SlaveGetDefaultConfig ( flexio_spi_slave_user_config_t * userConfigPtr )`

Returns default configuration structure for FLEXIO\_SPI slave.

#### Parameters

<i>userConfigPtr</i>	Pointer to the FLEXIO_SPI user configuration structure.
----------------------	---

Definition at line 1380 of file flexio\_spi\_driver.c.

**16.54.5.12** `status_t FLEXIO_SPI_DRV_SlaveGetStatus ( flexio_spi_slave_state_t * slave, uint32_t * bytesRemaining )`

Get the status of the current non-blocking SPI slave transaction.

This function returns the current status of a non-blocking SPI slave transaction. A return code of STATUS\_BUSY means the transfer is still in progress. Otherwise the function returns a status reflecting the outcome of the last transfer. When the driver is initialized in polling mode this function also advances the transfer by checking and handling the transmit and receive events, so it must be called frequently to avoid overflows or underflows.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_SPI slave driver context structure.
<i>bytesRemaining</i>	the remaining number of bytes to be transferred

**Returns**

Error or success status returned by API

This function returns the current status of a non-blocking SPI slave transaction. A return code of STATUS\_BUSY means the transfer is still in progress. Otherwise the function returns a status reflecting the outcome of the last transfer. When the driver is initialized in polling mode this function also advances the transfer by checking and handling the transmit and receive events, so it must be called frequently to avoid overflows or underflows.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_SPI slave driver context structure.
<i>bytesRemaining</i>	the remaining number of bytes to be transferred

**Returns**

Error or success status returned by API Implements : FLEXIO\_SPI\_DRV\_SlaveGetStatus\_Activity

Definition at line 1498 of file flexio\_spi\_driver.c.

**16.54.5.13** `status_t FLEXIO_SPI_DRV_SlaveInit ( uint32_t instance, const flexio_spi_slave_user_config_t * userConfigPtr, flexio_spi_slave_state_t * slave )`

Initialize the FLEXIO\_SPI slave mode driver.

This function initializes the FLEXIO\_SPI driver in slave mode.

**Parameters**

<i>instance</i>	FLEXIO peripheral instance number
<i>userConfigPtr</i>	Pointer to the FLEXIO_SPI slave user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
<i>slave</i>	Pointer to the FLEXIO_SPI slave driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using <a href="#">FLEXIO_SPI_DRV_SlaveDeinit()</a> .

**Returns**

Error or success status returned by API

Definition at line 1273 of file flexio\_spi\_driver.c.

**16.54.5.14** `status_t FLEXIO_SPI_DRV_SlaveTransfer ( flexio_spi_slave_state_t * slave, const uint8_t * txData, uint8_t * rxData, uint32_t dataSize )`

Perform a non-blocking SPI slave transaction.

This function performs an SPI full-duplex transaction, transmit and receive in parallel. If only transmit or receive are required, it is possible to provide NULL pointers for txData or rxData. The transfer is non-blocking, the function only initiates the transfer and then returns, leaving the transfer to complete asynchronously). [FLEXIO\\_SPI\\_DRV\\_SlaveGetStatus\(\)](#) can be called to check the status of the transfer.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_SPI slave driver context structure.
<i>txData</i>	pointer to the data to be transmitted
<i>rxData</i>	pointer to the buffer where to store received data
<i>dataSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API

This function performs an SPI full-duplex transaction, transmit and receive in parallel. If only transmit or receive are required, it is possible to provide NULL pointers for txData or rxData. The transfer is non-blocking, the function only initiates the transfer and then returns, leaving the transfer to complete asynchronously). [FLEXIO\\_SPI\\_DRV\\_SlaveGetStatus\(\)](#) can be called to check the status of the transfer.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_SPI slave driver context structure.
<i>txData</i>	pointer to the data to be transmitted
<i>rxData</i>	pointer to the buffer where to store received data
<i>dataSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API Implements : FLEXIO\_SPI\_DRV\_SlaveTransfer\_Activity

Definition at line 1433 of file flexio\_spi\_driver.c.

**16.54.5.15** `status_t FLEXIO_SPI_DRV_SlaveTransferAbort ( flexio_spi_slave_state_t * slave )`

Aborts a non-blocking SPI slave transaction.

This function aborts a non-blocking SPI transfer.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_SPI slave driver context structure.
--------------	---

**Returns**

Error or success status returned by API

This function aborts a non-blocking SPI transfer.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_SPI slave driver context structure.
--------------	---

**Returns**

Error or success status returned by API Implements : FLEXIO\_SPI\_DRV\_SlaveTransferAbort\_Activity

Definition at line 1477 of file flexio\_spi\_driver.c.

**16.54.5.16** `status_t FLEXIO_SPI_DRV_SlaveTransferBlocking ( flexio_spi_slave_state_t * slave, const uint8_t * txData, uint8_t * rxData, uint32_t dataSize, uint32_t timeout )`

Perform a blocking SPI slave transaction.

This function performs an SPI full-duplex transaction, transmit and receive in parallel. If only transmit or receive are required, it is possible to provide NULL pointers for txData or rxData. The transfer is blocking, the function only returns when the transfer is complete.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_SPI slave driver context structure.
<i>txData</i>	pointer to the data to be transmitted
<i>rxData</i>	pointer to the buffer where to store received data
<i>dataSize</i>	length in bytes of the data to be transferred
<i>timeout</i>	timeout for the transfer in milliseconds

**Returns**

Error or success status returned by API

This function performs an SPI full-duplex transaction, transmit and receive in parallel. If only transmit or receive are required, it is possible to provide NULL pointers for txData or rxData. The transfer is blocking, the function only returns when the transfer is complete.

**Parameters**

<i>slave</i>	Pointer to the FLEXIO_SPI slave driver context structure.
<i>txData</i>	pointer to the data to be transmitted
<i>rxData</i>	pointer to the buffer where to store received data
<i>dataSize</i>	length in bytes of the data to be transferred
<i>timeout</i>	timeout for the transfer in milliseconds

**Returns**

Error or success status returned by API Implements : FLEXIO\_SPI\_DRV\_SlaveTransferBlocking\_Activity

Definition at line 1458 of file flexio\_spi\_driver.c.



## 16.55 FlexIO UART Driver

### 16.55.1 Detailed Description

UART communication over FlexIO module (FLEXIO\_UART)

The FLEXIO\_UART Driver allows UART communication using the FlexIO module in the S32K1xx processors.

#### Features

- Interrupt, DMA or polling mode
- Provides blocking and non-blocking transmit and receive functions
- Configurable baud rate and number of bits
- Single stop bit only
- Parity bit not supported

#### Functionality

##### Initialization

Before using any Flexio driver the device must first be initialized using function `FLEXIO_DRV_InitDevice`. Then the FLEXIO\_UART Driver must be initialized, using function `FLEXIO_UART_DRV_Init()`. It is possible to use more driver instances on the same FlexIO device, as long as sufficient resources are available. Different driver instances on the same FlexIO device can function independently of each other. When it is no longer needed, the driver can be de-initialized, using `FLEXIO_UART_DRV_Deinit()`. This will release the hardware resources, allowing other driver instances to be initialized.

##### Choosing transmit/receive mode

To initialize the UART driver in transmit / receive mode the `direction` field of the configuration structure must be set to `FLEXIO_UART_DIRECTION_TX` / `FLEXIO_UART_DIRECTION_RX` when calling `FLEXIO_UART_DRV_Init()`. Once configured for one direction the driver must be used only for the chosen direction until it is de-initialized. One driver instance can only work in one direction at a time, but more driver instances can be created on the same device, up to the number of shifters present on the device (for example on S32K144 up to 4 driver instances can run in parallel on one device).

##### Setting the baud rate and bit count

The baud rate and bit count are provided at initialization time through the master configuration structure, but they can be changed at runtime by using function `FLEXIO_UART_DRV_SetConfig()`. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call `FLEXIO_UART_DRV_GetBaudRate()` to check what baud rate was actually set.

##### Transmitting / Receiving

To send or receive data to/from the currently configured slave address, use functions `FLEXIO_UART_DRV_SendData()` or `FLEXIO_UART_DRV_ReceiveData()` (or their blocking counterparts). Continuous send/receive can be realized by registering a user callback function. When the driver completes the transmission or reception of the current buffer, it will invoke the user callback with an appropriate event. The callback function can use `FLEXIO_UART_DRV_SetTxBuffer()` or `FLEXIO_UART_DRV_SetRxBuffer()` to provide a new buffer.

Blocking operations will return only when the transfer is completed, either successfully or with error. Non-blocking operations will initiate the transfer and return `STATUS_SUCCESS`, but the module is still busy with the transfer and another transfer can't be initiated until the current transfer is complete. The application will be notified through the user callback when the transfer completes, or it can check the status of the current transfer by calling `FLEXIO_UART_DRV_GetStatus()`. If the transfer is still ongoing this function will return `STATUS_BUSY`. If the transfer is

completed, the function will return either STATUS\_SUCCESS or an error code, depending on the outcome of the last transfer.

The driver supports interrupt, DMA and polling mode. In polling mode the function [FLEXIO\\_UART\\_DRV\\_GetStatus\(\)](#) ensures the progress of the transfer by checking and handling transmit and receive events reported by the FlexIO module. The application should ensure that this function is called often enough (at least once per transferred byte) to avoid Tx underflows or Rx overflows. In DMA mode the DMA channel that will be used by the driver is received through the configuration structure. The channel must be initialized by the application before the flexio\_↵uart driver is initialized. The flexio\_uart driver will only set the DMA request source.

#### Important Notes

- Before using the FLEXIO\_UART Driver the FlexIO clock must be configured. Refer to Clock Manager for clock configuration.
- Before using the FLEXIO\_UART Driver the pins must be routed to the FlexIO module. Refer to PINS Driver for pin routing configuration. Note that any of the available FlexIO pins can be used for the UART TX / RX line (configurable at initialization time). If more than one driver instance is used on the same Flexio module, it is the responsibility of the application to ensure there are no conflicts between pins.
- The driver enables the interrupts for the corresponding FlexIO module, but any interrupt priority setting must be done by the application.
- Timeout feature for blocking transfers does not work in polling mode.
- This driver needs one shifter and one timer for its operation. Initialization will fail if there are not enough shifters and timers available on the FlexIO device.
- This driver needs one DMA channel for its operation when it is initialized in DMA mode. The DMA channels must be initialized by the application before initializing the driver. Refer to EDMA driver for DMA channels initialization.
- If the application uses an RTOS, this driver uses a semaphore for blocking transfers. Initialization will fail if the semaphore cannot be created. If the driver uses polling mode no semaphore is used.
- If the application uses an RTOS, the FlexIO drivers use a mutex for channel allocation. Only one mutex per device is needed, not per driver instance. Device initialization will fail if the mutex cannot be created.
- For transfers where the data size is 2 bytes (bitCount is greater than 8) the driver assumes that the data buffers are defined with the proper type (uint16\_t) and are properly aligned.

#### Integration guideline

##### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\drivers\src\flexio\flexio_common.c
${S32SDK_PATH}\platform\drivers\src\flexio\flexio_uart_driver.c
```

##### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc
${S32SDK_PATH}\platform\drivers\src\flexio
```

##### Compile symbols

No special symbols are required for this component

##### Dependencies

[Clock Manager OS Interface \(OSIF\)](#) [Interrupt Manager \(Interrupt\)](#) [Enhanced Direct Memory Access \(eDMA\)](#)

## Data Structures

- struct [flexio\\_uart\\_user\\_config\\_t](#)  
Driver configuration structure. [More...](#)
- struct [flexio\\_uart\\_state\\_t](#)  
Driver internal context structure. [More...](#)

## Enumerations

- enum [flexio\\_uart\\_driver\\_direction\\_t](#) { [FLEXIO\\_UART\\_DIRECTION\\_TX](#) = 0x01U, [FLEXIO\\_UART\\_DIRECTION\\_RX](#) = 0x00U }  
*flexio\_uart driver direction (tx or rx)*

## FLEXIO\_UART Driver

- status\_t [FLEXIO\\_UART\\_DRV\\_Init](#) (uint32\_t instance, const [flexio\\_uart\\_user\\_config\\_t](#) \*userConfigPtr, [flexio\\_uart\\_state\\_t](#) \*state)  
*Initialize the FLEXIO\_UART driver.*
- status\_t [FLEXIO\\_UART\\_DRV\\_Deinit](#) ([flexio\\_uart\\_state\\_t](#) \*state)  
*De-initialize the FLEXIO\_UART driver.*
- status\_t [FLEXIO\\_UART\\_DRV\\_SetConfig](#) ([flexio\\_uart\\_state\\_t](#) \*state, uint32\_t baudRate, uint8\_t bitCount)  
*Set the baud rate and bit width for any subsequent UART communication.*
- status\_t [FLEXIO\\_UART\\_DRV\\_GetBaudRate](#) ([flexio\\_uart\\_state\\_t](#) \*state, uint32\_t \*baudRate)  
*Get the currently configured baud rate.*
- status\_t [FLEXIO\\_UART\\_DRV\\_SendDataBlocking](#) ([flexio\\_uart\\_state\\_t](#) \*state, const uint8\_t \*txBuff, uint32\_t txSize, uint32\_t timeout)  
*Perform a blocking UART transmission.*
- status\_t [FLEXIO\\_UART\\_DRV\\_SendData](#) ([flexio\\_uart\\_state\\_t](#) \*state, const uint8\_t \*txBuff, uint32\_t txSize)  
*Perform a non-blocking UART transmission.*
- status\_t [FLEXIO\\_UART\\_DRV\\_ReceiveDataBlocking](#) ([flexio\\_uart\\_state\\_t](#) \*state, uint8\_t \*rxBuff, uint32\_t rxSize, uint32\_t timeout)  
*Perform a blocking UART reception.*
- status\_t [FLEXIO\\_UART\\_DRV\\_ReceiveData](#) ([flexio\\_uart\\_state\\_t](#) \*state, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Perform a non-blocking UART reception.*
- status\_t [FLEXIO\\_UART\\_DRV\\_GetStatus](#) ([flexio\\_uart\\_state\\_t](#) \*state, uint32\_t \*bytesRemaining)  
*Get the status of the current non-blocking UART transfer.*
- status\_t [FLEXIO\\_UART\\_DRV\\_TransferAbort](#) ([flexio\\_uart\\_state\\_t](#) \*state)  
*Aborts a non-blocking UART transfer.*
- status\_t [FLEXIO\\_UART\\_DRV\\_SetRxBuffer](#) ([flexio\\_uart\\_state\\_t](#) \*state, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Provide a buffer for receiving data.*
- status\_t [FLEXIO\\_UART\\_DRV\\_SetTxBuffer](#) ([flexio\\_uart\\_state\\_t](#) \*state, const uint8\_t \*txBuff, uint32\_t txSize)  
*Provide a buffer for transmitting data.*
- void [FLEXIO\\_UART\\_DRV\\_GetDefaultConfig](#) ([flexio\\_uart\\_user\\_config\\_t](#) \*userConfigPtr)  
*Returns default configuration structure for FLEXIO\_UART.*

## 16.55.2 Data Structure Documentation

## 16.55.2.1 struct flexio\_uart\_user\_config\_t

Driver configuration structure.

This structure is used to provide configuration parameters for the flexio\_uart driver at initialization time. Implements : [flexio\\_uart\\_user\\_config\\_t\\_Class](#)

Definition at line 60 of file flexio\_uart\_driver.h.

## Data Fields

- [flexio\\_driver\\_type\\_t driverType](#)
- [uint32\\_t baudRate](#)
- [uint8\\_t bitCount](#)
- [flexio\\_uart\\_driver\\_direction\\_t direction](#)
- [uint8\\_t dataPin](#)
- [uart\\_callback\\_t callback](#)
- [void \\* callbackParam](#)
- [uint8\\_t dmaChannel](#)

## Field Documentation

### 16.55.2.1.1 [uint32\\_t baudRate](#)

Baud rate in hertz

Definition at line 63 of file [flexio\\_uart\\_driver.h](#).

### 16.55.2.1.2 [uint8\\_t bitCount](#)

Number of bits per word

Definition at line 64 of file [flexio\\_uart\\_driver.h](#).

### 16.55.2.1.3 [uart\\_callback\\_t callback](#)

User callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if it is not needed

Definition at line 67 of file [flexio\\_uart\\_driver.h](#).

### 16.55.2.1.4 [void\\* callbackParam](#)

Parameter for the callback function

Definition at line 71 of file [flexio\\_uart\\_driver.h](#).

### 16.55.2.1.5 [uint8\\_t dataPin](#)

Flexio pin to use as Tx or Rx pin

Definition at line 66 of file [flexio\\_uart\\_driver.h](#).

### 16.55.2.1.6 [flexio\\_uart\\_driver\\_direction\\_t direction](#)

Driver direction: Tx or Rx

Definition at line 65 of file [flexio\\_uart\\_driver.h](#).

### 16.55.2.1.7 [uint8\\_t dmaChannel](#)

DMA channel number. Only used in DMA mode

Definition at line 72 of file [flexio\\_uart\\_driver.h](#).

### 16.55.2.1.8 [flexio\\_driver\\_type\\_t driverType](#)

Driver type: interrupts/polling/DMA

Definition at line 62 of file [flexio\\_uart\\_driver.h](#).

### 16.55.2.2 [struct flexio\\_uart\\_state\\_t](#)

Driver internal context structure.

This structure is used by the flexio\_uart driver for its internal logic. It must be provided by the application through the `FLEXIO_UART_DRV_Init()` function, then it cannot be freed until the driver is de-initialized using `FLEXIO_UART_DRV_Deinit()`. The application should make no assumptions about the content of this structure.

Definition at line 84 of file flexio\_uart\_driver.h.

### 16.55.3 Enumeration Type Documentation

#### 16.55.3.1 enum flexio\_uart\_driver\_direction\_t

flexio\_uart driver direction (tx or rx)

This structure describes the direction configuration options for the flexio\_uart driver. Implements : flexio\_uart\_driver\_direction\_t\_Class

#### Enumerator

***FLEXIO\_UART\_DIRECTION\_TX*** Tx UART driver

***FLEXIO\_UART\_DIRECTION\_RX*** Rx UART driver

Definition at line 42 of file flexio\_uart\_driver.h.

### 16.55.4 Function Documentation

#### 16.55.4.1 status\_t FLEXIO\_UART\_DRV\_Deinit ( flexio\_uart\_state\_t \* state )

De-initialize the FLEXIO\_UART driver.

This function de-initializes the FLEXIO\_UART driver. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

#### Parameters

<i>state</i>	Pointer to the FLEXIO_UART driver context structure.
--------------	--

#### Returns

Error or success status returned by API

Definition at line 1065 of file flexio\_uart\_driver.c.

#### 16.55.4.2 status\_t FLEXIO\_UART\_DRV\_GetBaudRate ( flexio\_uart\_state\_t \* state, uint32\_t \* baudRate )

Get the currently configured baud rate.

This function returns the currently configured UART baud rate.

#### Parameters

<i>state</i>	Pointer to the FLEXIO_UART driver context structure.
<i>baudRate</i>	the current baud rate in hertz

#### Returns

Error or success status returned by API

Definition at line 1142 of file flexio\_uart\_driver.c.

#### 16.55.4.3 void FLEXIO\_UART\_DRV\_GetDefaultConfig ( flexio\_uart\_user\_config\_t \* userConfigPtr )

Returns default configuration structure for FLEXIO\_UART.

**Parameters**

<i>userConfigPtr</i>	Pointer to the FLEXIO_UART user configuration structure.
----------------------	--

Definition at line 1495 of file flexio\_uart\_driver.c.

**16.55.4.4** `status_t FLEXIO_UART_DRV_GetStatus ( flexio_uart_state_t * state, uint32_t * bytesRemaining )`

Get the status of the current non-blocking UART transfer.

This function returns the current status of a non-blocking UART transfer. A return code of STATUS\_BUSY means the transfer is still in progress. Otherwise the function returns a status reflecting the outcome of the last transfer. When the driver is initialized in polling mode this function also advances the transfer by checking and handling the transmit and receive events, so it must be called frequently to avoid overflows or underflows.

**Parameters**

<i>state</i>	Pointer to the FLEXIO_UART driver context structure.
<i>bytesRemaining</i>	the remaining number of bytes to be transferred

**Note**

In DMA mode, this parameter may not be accurate, in case the transfer completes right after calling this function; in this edge-case, the parameter will reflect the initial transfer size, due to automatic reloading of the major loop count in the DMA transfer descriptor.

**Returns**

Error or success status returned by API

Definition at line 1396 of file flexio\_uart\_driver.c.

**16.55.4.5** `status_t FLEXIO_UART_DRV_Init ( uint32_t instance, const flexio_uart_user_config_t * userConfigPtr, flexio_uart_state_t * state )`

Initialize the FLEXIO\_UART driver.

This function initializes the FLEXIO\_UART driver.

**Parameters**

<i>instance</i>	FLEXIO peripheral instance number
<i>userConfigPtr</i>	Pointer to the FLEXIO_UART user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
<i>state</i>	Pointer to the FLEXIO_UART driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using <a href="#">FLEXIO_UART_DRV_Deinit()</a> .

**Returns**

Error or success status returned by API

Definition at line 959 of file flexio\_uart\_driver.c.

**16.55.4.6** `status_t FLEXIO_UART_DRV_ReceiveData ( flexio_uart_state_t * state, uint8_t * rxBuff, uint32_t rxSize )`

Perform a non-blocking UART reception.

This function receives a block of data and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the [FLEXIO\\_UART\\_DRV\\_GetReceiveStatus\(\)](#) function (if the driver is initialized in polling mode).

## Parameters

<i>state</i>	Pointer to the FLEXIO_UART driver context structure.
<i>rxBuff</i>	pointer to the receive buffer
<i>rxSize</i>	length in bytes of the data to be received

## Returns

Error or success status returned by API

Definition at line 1278 of file flexio\_uart\_driver.c.

**16.55.4.7** `status_t FLEXIO_UART_DRV_ReceiveDataBlocking ( flexio_uart_state_t * state, uint8_t * rxBuff, uint32_t rxSize, uint32_t timeout )`

Perform a blocking UART reception.

This function receives a block of data and only returns when the transmission is complete.

## Parameters

<i>state</i>	Pointer to the FLEXIO_UART driver context structure.
<i>rxBuff</i>	pointer to the receive buffer
<i>rxSize</i>	length in bytes of the data to be received
<i>timeout</i>	timeout for the transfer in milliseconds

## Returns

Error or success status returned by API

Definition at line 1339 of file flexio\_uart\_driver.c.

**16.55.4.8** `status_t FLEXIO_UART_DRV_SendData ( flexio_uart_state_t * state, const uint8_t * txBuff, uint32_t txSize )`

Perform a non-blocking UART transmission.

This function sends a block of data and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode) or by the FLEXIO\_UART\_DRV\_GetTransmitStatus() function (if the driver is initialized in polling mode).

## Parameters

<i>state</i>	Pointer to the FLEXIO_UART driver context structure.
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred

## Returns

Error or success status returned by API

Definition at line 1182 of file flexio\_uart\_driver.c.

**16.55.4.9** `status_t FLEXIO_UART_DRV_SendDataBlocking ( flexio_uart_state_t * state, const uint8_t * txBuff, uint32_t txSize, uint32_t timeout )`

Perform a blocking UART transmission.

This function sends a block of data and only returns when the transmission is complete.

**Parameters**

<i>state</i>	Pointer to the FLEXIO_UART driver context structure.
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred
<i>timeout</i>	timeout for the transfer in milliseconds

**Returns**

Error or success status returned by API

Definition at line 1245 of file flexio\_uart\_driver.c.

**16.55.4.10** `status_t FLEXIO_UART_DRV_SetConfig ( flexio_uart_state_t * state, uint32_t baudRate, uint8_t bitCount )`

Set the baud rate and bit width for any subsequent UART communication.

This function sets the baud rate and bit width for the UART driver. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency FlexIO clock. The application should call [FLEXIO\\_UART\\_DRV\\_GetBaudRate\(\)](#) after [FLEXIO\\_UART\\_DRV\\_SetConfig\(\)](#) to check what baud rate was actually set.

**Parameters**

<i>state</i>	Pointer to the FLEXIO_UART driver context structure.
<i>baudRate</i>	the desired baud rate in hertz
<i>bitCount</i>	number of bits per word

**Returns**

Error or success status returned by API

Definition at line 1092 of file flexio\_uart\_driver.c.

**16.55.4.11** `status_t FLEXIO_UART_DRV_SetRxBuffer ( flexio_uart_state_t * state, uint8_t * rxBuff, uint32_t rxSize )`

Provide a buffer for receiving data.

This function can be used to provide a new buffer for receiving data to the driver. It can be called from the user callback when event STATUS\_UART\_RX\_OVERRUN is reported. This way the reception will continue without interruption.

**Parameters**

<i>state</i>	Pointer to the FLEXIO_UART driver context structure.
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API

Definition at line 1449 of file flexio\_uart\_driver.c.

**16.55.4.12** `status_t FLEXIO_UART_DRV_SetTxBuffer ( flexio_uart_state_t * state, const uint8_t * txBuff, uint32_t txSize )`

Provide a buffer for transmitting data.

This function can be used to provide a new buffer for transmitting data to the driver. It can be called from the user callback when event STATUS\_UART\_TX\_UNDERRUN is reported. This way the transmission will continue without interruption.



**Parameters**

<i>state</i>	Pointer to the FLEXIO_UART driver context structure.
<i>txBuff</i>	pointer to the buffer containing transmit data
<i>txSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API

Definition at line 1471 of file flexio\_uart\_driver.c.

**16.55.4.13** `status_t FLEXIO_UART_DRV_TransferAbort( flexio_uart_state_t * state )`

Aborts a non-blocking UART transfer.

This function aborts a non-blocking UART transfer.

**Parameters**

<i>state</i>	Pointer to the FLEXIO_UART driver context structure.
--------------	--

**Returns**

Error or success status returned by API

Definition at line 1372 of file flexio\_uart\_driver.c.

## 16.56 FlexTimer (FTM)

### 16.56.1 Detailed Description

FlexTimer Peripheral Driver.

#### Hardware background

The FTM of the S32K1xx is based on a 16 bits counter and supports: input capture, output compare, PWM and some instances include quadrature decoder. The main features are:

- FTM source clock is selectable (Source clock can be the system clock, the fixed frequency clock, or an external clock)
- Prescaler: 1, 2, 4, 8, 16, 32, 64, 128
- 16 bit counter (up and up-down counting)
- Each channel can be configured for input capture, output compare, or PWM mode.
- Input Capture mode (single edge, dual edge or measure period/duty cycle)
- Output Compare mode (set, cleared or toggle on match)
- All channels can be configured for center-aligned PWM mode.
- Each pair of channels can be combined to generate a PWM signal with independent control of both edges of PWM signal and with dead-time insertion.
- Up to 4 fault inputs for global fault control
- Dual edge capture for pulse and period width measurement
- Quadrature decoder with input filters, relative position counting, and interrupt on position count or capture of position count on external event.

#### Modules

- [FlexTimer Input Capture Driver \(FTM\\_IC\)](#)  
*FlexTimer Input Capture Peripheral Driver.*
- [FlexTimer Module Counter Driver \(FTM\\_MC\)](#)  
*FlexTimer Module Counter Peripheral Driver.*
- [FlexTimer Output Compare Driver \(FTM\\_OC\)](#)  
*FlexTimer Output Compare Peripheral Driver.*
- [FlexTimer Pulse Width Modulation Driver \(FTM\\_PWM\)](#)  
*FlexTimer Pulse Width Modulation Peripheral Driver.*
- [FlexTimer Quadrature Decoder Driver \(FTM\\_QD\)](#)  
*FlexTimer Quadrature Decoder Peripheral Driver.*

#### Data Structures

- struct [ftm\\_state\\_t](#)  
*FlexTimer state structure of the driver. [More...](#)*
- struct [ftm\\_pwm\\_sync\\_t](#)  
*FlexTimer Registers sync parameters Please don't use software and hardware trigger simultaneously Implements : [ftm\\_pwm\\_sync\\_t\\_Class](#). [More...](#)*
- struct [ftm\\_user\\_config\\_t](#)  
*Configuration structure that the user needs to set. [More...](#)*

## Macros

- #define **FTM\_RMW\_SC**(base, mask, value) (((base)->SC) = (((base)->SC) & ~(mask)) | (value)))  
*FTM\_SC - Read and modify and write to Status And Control (RW)*
- #define **FTM\_RMW\_CNT**(base, mask, value) (((base)->CNT) = (((base)->CNT) & ~(mask)) | (value)))  
*FTM\_CNT - Read and modify and write to Counter (RW)*
- #define **FTM\_RMW\_MOD**(base, mask, value) (((base)->MOD) = (((base)->MOD) & ~(mask)) | (value)))  
*FTM\_MOD - Read and modify and write Modulo (RW)*
- #define **FTM\_RMW\_CNTIN**(base, mask, value) (((base)->CNTIN) = (((base)->CNTIN) & ~(mask)) | (value)))  
*FTM\_CNTIN - Read and modify and write Counter Initial Value (RW)*
- #define **FTM\_RMW\_STATUS**(base, mask, value) (((base)->STATUS) = (((base)->STATUS) & ~(mask)) | (value)))  
*FTM\_STATUS - Read and modify and write Capture And Compare Status (RW)*
- #define **FTM\_RMW\_MODE**(base, mask, value) (((base)->MODE) = (((base)->MODE) & ~(mask)) | (value)))  
*FTM\_MODE - Read and modify and write Counter Features Mode Selection (RW)*
- #define **FTM\_RMW\_CnSCV\_REG**(base, channel, mask, value) (((base)->CONTROLS[channel].CnSC) = (((base)->CONTROLS[channel].CnSC) & ~(mask)) | (value)))  
*FTM\_CnSCV - Read and modify and write Channel (n) Status And Control (RW)*
- #define **FTM\_RMW\_DEADTIME**(base, mask, value) (((base)->DEADTIME) = (((base)->DEADTIME) & ~(mask)) | (value)))  
*FTM\_DEADTIME - Read and modify and write Dead-time Insertion Control (RW)*
- #define **FTM\_RMW\_EXTTRIG\_REG**(base, mask, value) (((base)->EXTTRIG) = (((base)->EXTTRIG) & ~(mask)) | (value)))  
*FTM\_EXTTRIG - Read and modify and write External Trigger Control (RW)*
- #define **FTM\_RMW\_FLTCTRL**(base, mask, value) (((base)->FLTCTRL) = (((base)->FLTCTRL) & ~(mask)) | (value)))  
*FTM\_FLTCTRL - Read and modify and write Fault Control (RW)*
- #define **FTM\_RMW\_FMS**(base, mask, value) (((base)->FMS) = (((base)->FMS) & ~(mask)) | (value)))  
*FTM\_FMS - Read and modify and write Fault Mode Status (RW)*
- #define **FTM\_RMW\_CONF**(base, mask, value) (((base)->CONF) = (((base)->CONF) & ~(mask)) | (value)))  
*FTM\_CONF - Read and modify and write Configuration (RW)*
- #define **FTM\_RMW\_POL**(base, mask, value) (((base)->POL) = (((base)->POL) & ~(mask)) | (value)))  
*POL - Read and modify and write Polarity (RW)*
- #define **FTM\_RMW\_FILTER**(base, mask, value) (((base)->FILTER) = (((base)->FILTER) & ~(mask)) | (value)))  
*FILTER - Read and modify and write Filter (RW)*
- #define **FTM\_RMW\_SYNC**(base, mask, value) (((base)->SYNC) = (((base)->SYNC) & ~(mask)) | (value)))  
*SYNC - Read and modify and write Synchronization (RW)*
- #define **FTM\_RMW\_QDCTRL**(base, mask, value) (((base)->QDCTRL) = (((base)->QDCTRL) & ~(mask)) | (value)))  
*QDCTRL - Read and modify and write Quadrature Decoder Control And Status (RW)*
- #define **FTM\_RMW\_PAIR0DEADTIME**(base, mask, value) (((base)->PAIR0DEADTIME) = (((base)->PAIR0DEADTIME) & ~(mask)) | (value)))  
*FTM\_PAIR0DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 0 (RW)*
- #define **FTM\_RMW\_PAIR1DEADTIME**(base, mask, value) (((base)->PAIR1DEADTIME) = (((base)->PAIR1DEADTIME) & ~(mask)) | (value)))  
*FTM\_PAIR1DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 1 (RW)*
- #define **FTM\_RMW\_PAIR2DEADTIME**(base, mask, value) (((base)->PAIR2DEADTIME) = (((base)->PAIR2DEADTIME) & ~(mask)) | (value)))  
*FTM\_PAIR2DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 2 (RW)*

- #define `FTM_RMW_PAIR3DEADTIME`(base, mask, value) (((base)->PAIR3DEADTIME) = (((base)->PAIR3DEADTIME) & ~(mask)) | (value)))  
*FTM\_PAIR3DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 3 (RW)*
- #define `CHAN0_IDX` (0U)  
*Channel number for CHAN1.*
- #define `CHAN1_IDX` (1U)  
*Channel number for CHAN2.*
- #define `CHAN2_IDX` (2U)  
*Channel number for CHAN3.*
- #define `CHAN3_IDX` (3U)  
*Channel number for CHAN4.*
- #define `CHAN4_IDX` (4U)  
*Channel number for CHAN5.*
- #define `CHAN5_IDX` (5U)  
*Channel number for CHAN6.*
- #define `CHAN6_IDX` (6U)  
*Channel number for CHAN7.*
- #define `CHAN7_IDX` (7U)

## Enumerations

- enum `ftm_config_mode_t` {  
`FTM_MODE_NOT_INITIALIZED` = 0x00U, `FTM_MODE_INPUT_CAPTURE` = 0x01U, `FTM_MODE_OUTPUT_COMPARE` = 0x02U, `FTM_MODE_EDGE_ALIGNED_PWM` = 0x03U,  
`FTM_MODE_CEN_ALIGNED_PWM` = 0x04U, `FTM_MODE_QUADRATURE_DECODER` = 0x05U, `FTM_MODE_UP_TIMER` = 0x06U, `FTM_MODE_UP_DOWN_TIMER` = 0x07U,  
`FTM_MODE_EDGE_ALIGNED_PWM_AND_INPUT_CAPTURE` = 0x08U }  
*FlexTimer operation mode.*
- enum `ftm_clock_source_t` { `FTM_CLOCK_SOURCE_NONE` = 0x00U, `FTM_CLOCK_SOURCE_SYSTEM_CLK` = 0x01U, `FTM_CLOCK_SOURCE_FIXEDCLK` = 0x02U, `FTM_CLOCK_SOURCE_EXTERNALCLK` = 0x03U }  
*FlexTimer clock source selection.*
- enum `ftm_clock_ps_t` {  
`FTM_CLOCK_DIVID_BY_1` = 0x00U, `FTM_CLOCK_DIVID_BY_2` = 0x01U, `FTM_CLOCK_DIVID_BY_4` = 0x02U, `FTM_CLOCK_DIVID_BY_8` = 0x03U,  
`FTM_CLOCK_DIVID_BY_16` = 0x04U, `FTM_CLOCK_DIVID_BY_32` = 0x05U, `FTM_CLOCK_DIVID_BY_64` = 0x06U, `FTM_CLOCK_DIVID_BY_128` = 0x07U }  
*FlexTimer pre-scaler factor selection for the clock source. In quadrature decoder mode set FTM\_CLOCK\_DIVID\_BY\_1.*
- enum `ftm_interrupt_option_t` {  
`FTM_CHANNEL0_INT_ENABLE` = 0x00000001U, `FTM_CHANNEL1_INT_ENABLE` = 0x00000002U, `FTM_CHANNEL2_INT_ENABLE` = 0x00000004U, `FTM_CHANNEL3_INT_ENABLE` = 0x00000008U,  
`FTM_CHANNEL4_INT_ENABLE` = 0x00000010U, `FTM_CHANNEL5_INT_ENABLE` = 0x00000020U, `FTM_CHANNEL6_INT_ENABLE` = 0x00000040U, `FTM_CHANNEL7_INT_ENABLE` = 0x00000080U,  
`FTM_FAULT_INT_ENABLE` = 0x00000100U, `FTM_TIME_OVER_FLOW_INT_ENABLE` = 0x00000200U, `FTM_RELOAD_INT_ENABLE` = 0x00000400U }  
*List of FTM interrupts.*
- enum `ftm_status_flag_t` {  
`FTM_CHANNEL0_FLAG` = 0x00000001U, `FTM_CHANNEL1_FLAG` = 0x00000002U, `FTM_CHANNEL2_FLAG` = 0x00000004U, `FTM_CHANNEL3_FLAG` = 0x00000008U,  
`FTM_CHANNEL4_FLAG` = 0x00000010U, `FTM_CHANNEL5_FLAG` = 0x00000020U, `FTM_CHANNEL6_FLAG` = 0x00000040U, `FTM_CHANNEL7_FLAG` = 0x00000080U,  
`FTM_FAULT_FLAG` = 0x00000100U, `FTM_TIME_OVER_FLOW_FLAG` = 0x00000200U, `FTM_RELOAD_FLAG` = 0x00000400U, `FTM_CHANNEL_TRIGGER_FLAG` = 0x00000800U }

List of FTM flags.

- enum `ftm_reg_update_t` { `FTM_SYSTEM_CLOCK` = 0U, `FTM_PWM_SYNC` = 1U }

FTM sync source.

- enum `ftm_pwm_sync_mode_t` { `FTM_WAIT_LOADING_POINTS` = 0U, `FTM_UPDATE_NOW` = 1U }

FTM update register.

- enum `ftm_deadtime_ps_t` { `FTM_DEADTIME_DIVID_BY_1` = 0x01U, `FTM_DEADTIME_DIVID_BY_4` = 0x02U, `FTM_DEADTIME_DIVID_BY_16` = 0x03U }

FlexTimer pre-scaler factor for the dead-time insertion.

- enum `ftm_bdm_mode_t` { `FTM_BDM_MODE_00` = 0x00U, `FTM_BDM_MODE_01` = 0x01U, `FTM_BDM_MODE_10` = 0x02U, `FTM_BDM_MODE_11` = 0x03U }

Options for the FlexTimer behavior in BDM Mode.

## Functions

- static void `FTM_DRV_SetClockFilterPs` (FTM\_Type \*const ftmBase, uint8\_t filterPrescale)  
*Sets the filter Pre-scaler divider.*
- static uint8\_t `FTM_DRV_GetClockFilterPs` (const FTM\_Type \*ftmBase)  
*Reads the FTM filter clock divider.*
- static uint16\_t `FTM_DRV_GetCounter` (const FTM\_Type \*ftmBase)  
*Returns the FTM peripheral current counter value.*
- static uint16\_t `FTM_DRV_GetMod` (const FTM\_Type \*ftmBase)  
*Returns the FTM peripheral counter modulo value.*
- static uint16\_t `FTM_DRV_GetCounterInitVal` (const FTM\_Type \*ftmBase)  
*Returns the FTM peripheral counter initial value.*
- static void `FTM_DRV_ClearChSC` (FTM\_Type \*const ftmBase, uint8\_t channel)  
*Clears the content of Channel (n) Status And Control.*
- static uint8\_t `FTM_DRV_GetChnEdgeLevel` (const FTM\_Type \*ftmBase, uint8\_t channel)  
*Gets the FTM peripheral timer channel edge level.*
- static void `FTM_DRV_SetChnIcrstCmd` (FTM\_Type \*const ftmBase, uint8\_t channel, bool enable)  
*Configure the feature of FTM counter reset by the selected input capture event.*
- static bool `FTM_DRV_IsChnIcrst` (const FTM\_Type \*ftmBase, uint8\_t channel)  
*Returns whether the FTM FTM counter is reset.*
- static void `FTM_DRV_SetChnDmaCmd` (FTM\_Type \*const ftmBase, uint8\_t channel, bool enable)  
*Enables or disables the FTM peripheral timer channel DMA.*
- static bool `FTM_DRV_IsChnDma` (const FTM\_Type \*ftmBase, uint8\_t channel)  
*Returns whether the FTM peripheral timer channel DMA is enabled.*
- static void `FTM_DRV_SetTrigModeControlCmd` (FTM\_Type \*const ftmBase, uint8\_t channel, bool enable)  
*Enables or disables the trigger generation on FTM channel outputs.*
- static bool `FTM_DRV_GetTriggerControlled` (const FTM\_Type \*ftmBase, uint8\_t channel)  
*Returns whether the trigger mode is enabled.*
- static bool `FTM_DRV_GetChnInputState` (const FTM\_Type \*ftmBase, uint8\_t channel)  
*Get the state of channel input.*
- static bool `FTM_DRV_GetChnOutputValue` (const FTM\_Type \*ftmBase, uint8\_t channel)  
*Get the value of channel output.*
- static uint16\_t `FTM_DRV_GetChnCountVal` (const FTM\_Type \*ftmBase, uint8\_t channel)  
*Gets the FTM peripheral timer channel counter value.*
- static bool `FTM_DRV_GetChnEventStatus` (const FTM\_Type \*ftmBase, uint8\_t channel)  
*Gets the FTM peripheral timer channel event status.*
- static uint32\_t `FTM_DRV_GetEventStatus` (const FTM\_Type \*ftmBase)  
*Gets the FTM peripheral timer status info for all channels.*
- static void `FTM_DRV_ClearChnEventStatus` (FTM\_Type \*const ftmBase, uint8\_t channel)

- Clears the FTM peripheral timer all channel event status.*
- static void [FTM\\_DRV\\_SetChnOutputMask](#) (FTM\_Type \*const ftmBase, uint8\_t channel, bool mask)  
*Sets the FTM peripheral timer channel output mask.*
- static void [FTM\\_DRV\\_SetChnOutputInitStateCmd](#) (FTM\_Type \*const ftmBase, uint8\_t channel, bool state)  
*Sets the FTM peripheral timer channel output initial state 0 or 1.*
- static void [FTM\\_DRV\\_DisableFaultInt](#) (FTM\_Type \*const ftmBase)  
*Disables the FTM peripheral timer fault interrupt.*
- static void [FTM\\_DRV\\_SetCaptureTestCmd](#) (FTM\_Type \*const ftmBase, bool enable)  
*Enables or disables the FTM peripheral timer capture test mode.*
- static bool [FTM\\_DRV\\_IsFtmEnable](#) (const FTM\_Type \*ftmBase)  
*Get status of the FTMEN bit in the FTM\_MODE register.*
- static void [FTM\\_DRV\\_SetCountReinitSyncCmd](#) (FTM\_Type \*const ftmBase, bool enable)  
*Determines if the FTM counter is re-initialized when the selected trigger for synchronization is detected.*
- static bool [FTM\\_DRV\\_IsWriteProtectionEnabled](#) (const FTM\_Type \*ftmBase)  
*Checks whether the write protection is enabled.*
- static bool [FTM\\_DRV\\_IsFaultInputEnabled](#) (const FTM\_Type \*ftmBase)  
*Checks whether the logic OR of the fault inputs is enabled.*
- static bool [FTM\\_DRV\\_IsFaultFlagDetected](#) (const FTM\_Type \*ftmBase, uint8\_t channel)  
*Checks whether a fault condition is detected at the fault input.*
- static void [FTM\\_DRV\\_ClearFaultFlagDetected](#) (FTM\_Type \*const ftmBase, uint8\_t channel)  
*Clear a fault condition is detected at the fault input.*
- static void [FTM\\_DRV\\_SetDualChnInvertCmd](#) (FTM\_Type \*const ftmBase, uint8\_t chnIPairNum, bool enable)  
*Enables or disables the channel invert for a channel pair.*
- static void [FTM\\_DRV\\_SetChnSoftwareCtrlCmd](#) (FTM\_Type \*const ftmBase, uint8\_t channel, bool enable)  
*Enables or disables the channel software output control.*
- static void [FTM\\_DRV\\_SetChnSoftwareCtrlVal](#) (FTM\_Type \*const ftmBase, uint8\_t channel, bool enable)  
*Sets the channel software output control value. Despite the odd channels are configured as HIGH/LOW, they will be inverted in the following configuration: COMP bit = 1 and CH(n)OCV and CH(n+1)OCV are HIGH. Please check Software output control behavior chapter from RM.*
- static void [FTM\\_DRV\\_SetGlobalLoadCmd](#) (FTM\_Type \*const ftmBase)  
*Set the global load mechanism.*
- static void [FTM\\_DRV\\_SetLoadCmd](#) (FTM\_Type \*const ftmBase, bool enable)  
*Enable the global load.*
- static void [FTM\\_DRV\\_SetHalfCycleCmd](#) (FTM\_Type \*const ftmBase, bool enable)  
*Enable the half cycle reload.*
- static void [FTM\\_DRV\\_SetPwmLoadCmd](#) (FTM\_Type \*const ftmBase, bool enable)  
*Enables or disables the loading of MOD, CNTIN and CV with values of their write buffer.*
- static void [FTM\\_DRV\\_SetPwmLoadChnSelCmd](#) (FTM\_Type \*const ftmBase, uint8\_t channel, bool enable)  
*Includes or excludes the channel in the matching process.*
- static void [FTM\\_DRV\\_SetInitTrigOnReloadCmd](#) (FTM\_Type \*const ftmBase, bool enable)  
*Enables or disables the FTM initialization trigger on Reload Point.*
- static void [FTM\\_DRV\\_SetGlobalTimeBaseOutputCmd](#) (FTM\_Type \*const ftmBase, bool enable)  
*Enables or disables the FTM global time base signal generation to other FTM's.*
- static void [FTM\\_DRV\\_SetGlobalTimeBaseCmd](#) (FTM\_Type \*const ftmBase, bool enable)  
*Enables or disables the FTM timer global time base.*
- static void [FTM\\_DRV\\_SetLoadFreq](#) (FTM\_Type \*const ftmBase, uint8\_t val)  
*Sets the frequency of reload points.*
- static void [FTM\\_DRV\\_SetExtPairDeadtimeValue](#) (FTM\_Type \*const ftmBase, uint8\_t channelPair, uint8\_t value)  
*Sets the FTM extended dead-time value for the channel pair.*

- static void `FTM_DRV_SetPairDeadtimePrescale` (FTM\_Type \*const ftmBase, uint8\_t channelPair, `ftm_deadtime_ps_t` divider)
 

*Sets the FTM dead time divider for the channel pair.*
- static void `FTM_DRV_SetPairDeadtimeCount` (FTM\_Type \*const ftmBase, uint8\_t channelPair, uint8\_t count)
 

*Sets the FTM dead-time value for the channel pair.*
- status\_t `FTM_DRV_Init` (uint32\_t instance, const `ftm_user_config_t` \*info, `ftm_state_t` \*state)
 

*Initializes the FTM driver.*
- status\_t `FTM_DRV_Deinit` (uint32\_t instance)
 

*Shuts down the FTM driver.*
- void `FTM_DRV_GetDefaultConfig` (`ftm_user_config_t` \*const config)
 

*This function will get the default configuration values in the structure which is used as a common use-case.*
- status\_t `FTM_DRV_MaskOutputChannels` (uint32\_t instance, uint32\_t channelsMask, bool softwareTrigger)
 

*This function will mask the output of the channels and at match events will be ignored by the masked channels.*
- status\_t `FTM_DRV_SetInitialCounterValue` (uint32\_t instance, uint16\_t counterValue, bool softwareTrigger)
 

*This function configure the initial counter value. The counter will get this value after an overflow event.*
- status\_t `FTM_DRV_SetHalfCycleReloadPoint` (uint32\_t instance, uint16\_t reloadPoint, bool softwareTrigger)
 

*This function configure the value of the counter which will generates an reload point.*
- status\_t `FTM_DRV_SetSoftOutChnValue` (uint32\_t instance, uint8\_t channelsValues, bool softwareTrigger)
 

*This function will force the output value of a channel to a specific value. Before using this function it's mandatory to mask the match events using `FTM_DRV_MaskOutputChannels` and to enable software output control using `FTM_DRV_SetSoftwareOutputChannelControl`. : When the PWM signal is configured with LOW/HIGH polarity on the channel (n). It should be set the safe state as LOW level state. However, We will have an issue with COMP bit is zero and CH(n)OCV is HIGH and CH(n+1)OCV is LOW.in the independent channel configuration. Code configuration: { .polarity = FTM\_POLARITY\_HIGH, .safeState = FTM\_POLARITY\_LOW, .enableSecondChannelOutput = true, .secondChannelPolarity = FTM\_MAIN\_DUPLICATED, }.*
- status\_t `FTM_DRV_SetSoftwareOutputChannelControl` (uint32\_t instance, uint8\_t channelsMask, bool softwareTrigger)
 

*This function will configure which output channel can be software controlled. Software output control forces the following values on channels (n) and (n+1) when the COMP bit is zero and POL bit is zero. CH(n)OCV|CH(n+1)OCV|CH(n)OCV|CH(n+1)OCV|Channel (n) Output | Channel (n+1) Output 0 | 0 | X | X | is not modified by SWOC | is not modified by SWOC 1 | 1 | 0 | 0 | is forced to zero | is forced to zero 1 | 1 | 0 | 1 | is forced to zero | is forced to one 1 | 1 | 1 | 0 | is forced to one | is forced to zero 1 | 1 | 1 | 1 | is forced to one | is forced to one.*
- status\_t `FTM_DRV_SetAllChnSoftwareOutputControl` (uint32\_t instance, uint8\_t channelMask, uint8\_t channelValueMask, bool softwareTrigger)
 

*This function will control list of channels by software to force the output to specified value. Despite the odd channels are configured as HIGH/LOW, they will be inverted in the following configuration: COMP bit = 1 and CH(n)OCV and CH(n+1)OCV are HIGH. Please check software output control behavior chapter from reference manual. : When the PWM signal is configured with LOW/HIGH polarity on the channel (n). It should be set the safe state as LOW level state. However, We will have an issue with COMP bit is zero and CH(n)OCV is HIGH and CH(n+1)OCV is LOW.in the independent channel configuration. Code configuration: { .polarity = FTM\_POLARITY\_HIGH, .safeState = FTM\_POLARITY\_LOW, .enableSecondChannelOutput = true, .secondChannelPolarity = FTM\_MAIN\_DUPLICATED, }.*
- status\_t `FTM_DRV_SetInvertingControl` (uint32\_t instance, uint8\_t channelsPairMask, bool softwareTrigger)
 

*This function will configure if the second channel of a pair will be inverted or not.*
- status\_t `FTM_DRV_SetModuloCounterValue` (uint32\_t instance, uint16\_t counterValue, bool softwareTrigger)
 

*This function configure the maximum counter value.*
- status\_t `FTM_DRV_SetOutputlevel` (uint32\_t instance, uint8\_t channel, uint8\_t level)
 

*This function will set the channel edge or level on the selection of the channel mode.*
- status\_t `FTM_DRV_SetSync` (uint32\_t instance, const `ftm_pwm_sync_t` \*param)
 

*This function configures sync mechanism for some FTM registers (MOD, CNINT, HCR, CnV, OUTMASK, INVCTRL, SWOCTRL).*
- status\_t `FTM_DRV_GenerateHardwareTrigger` (uint32\_t instance)
 

*This function is used to configure a trigger source for FTM instance. This allow a hardware trigger input which can be used in PWM synchronization. Note that the hardware trigger is implemented only on trigger 1 for each instance.*
- status\_t `FTM_DRV_EnableInterrupts` (uint32\_t instance, uint32\_t interruptMask)



*This function will enable the generation a list of interrupts. It includes the FTM overflow interrupts, the reload point interrupt, the fault interrupt and the channel (n) interrupt.*

- void [FTM\\_DRV\\_DisableInterrupts](#) (uint32\_t instance, uint32\_t interruptMask)

*This function is used to disable some interrupts.*

- uint32\_t [FTM\\_DRV\\_GetEnabledInterrupts](#) (uint32\_t instance)

*This function will get the enabled FTM interrupts.*

- uint32\_t [FTM\\_DRV\\_GetStatusFlags](#) (uint32\_t instance)

*This function will get the FTM status flags. : Regarding the duty cycle is 100% at the channel output, the match interrupt has no event due to the C(n)V and C(n+1)V value are not between CNTIN value and MOD value.*

- void [FTM\\_DRV\\_ClearStatusFlags](#) (uint32\_t instance, uint32\_t flagMask)

*This function is used to clear the FTM status flags.*

- uint32\_t [FTM\\_DRV\\_GetFrequency](#) (uint32\_t instance)

*Retrieves the frequency of the clock source feeding the FTM counter.*

- uint16\_t [FTM\\_DRV\\_ConvertFreqToPeriodTicks](#) (uint32\_t instance, uint32\_t frequencyHz)

*This function is used to covert the given frequency to period in ticks.*

- status\_t [FTM\\_DRV\\_CounterReset](#) (uint32\_t instance, bool softwareTrigger)

*This function will allow the FTM to restart the counter to its initial counting value in the register. Note that the configuration is set in the [FTM\\_DRV\\_SetSync\(\)](#) function to make sure that the FTM registers are updated by software trigger or hardware trigger.*

## Variables

- FTM\_Type \*const [g\\_ftmBase](#) [FTM\_INSTANCE\_COUNT]

*Table of base addresses for FTM instances.*

- const IRQn\_Type [g\\_ftmIrqlId](#) [FTM\_INSTANCE\_COUNT][FEATURE\_FTM\_CHANNEL\_COUNT]

*Interrupt vectors for the FTM peripheral.*

- const IRQn\_Type [g\\_ftmFaultIrqlId](#) [FTM\_INSTANCE\_COUNT]
- const IRQn\_Type [g\\_ftmOverflowIrqlId](#) [FTM\_INSTANCE\_COUNT]
- const IRQn\_Type [g\\_ftmReloadIrqlId](#) [FTM\_INSTANCE\_COUNT]
- [ftm\\_state\\_t](#) \* [ftmStatePtr](#) [FTM\_INSTANCE\_COUNT]

*Pointer to runtime state structure.*

## 16.56.2 Data Structure Documentation

### 16.56.2.1 struct ftm\_state\_t

FlexTimer state structure of the driver.

Implements : [ftm\\_state\\_t\\_Class](#)

Definition at line 376 of file [ftm\\_common.h](#).

## Data Fields

- [ftm\\_clock\\_source\\_t](#) [ftmClockSource](#)
- [ftm\\_config\\_mode\\_t](#) [ftmMode](#)
- uint16\_t [ftmModValue](#)
- uint16\_t [ftmPeriod](#)
- uint32\_t [ftmSourceClockFrequency](#)
- uint16\_t [measurementResults](#) [FEATURE\_FTM\_CHANNEL\_COUNT]
- void \* [channelsCallbacksParams](#) [FEATURE\_FTM\_CHANNEL\_COUNT]
- ic\_callback\_t [channelsCallbacks](#) [FEATURE\_FTM\_CHANNEL\_COUNT]
- bool [enableNotification](#) [FEATURE\_FTM\_CHANNEL\_COUNT]



## Field Documentation

### 16.56.2.1.1 `ic_callback_t channelsCallbacks[FEATURE_FTM_CHANNEL_COUNT]`

The callback function for channels events

Definition at line 385 of file `ftm_common.h`.

### 16.56.2.1.2 `void* channelsCallbacksParams[FEATURE_FTM_CHANNEL_COUNT]`

The parameters of callback function for channels events

Definition at line 384 of file `ftm_common.h`.

### 16.56.2.1.3 `bool enableNotification[FEATURE_FTM_CHANNEL_COUNT]`

To save channels enable the notification on the callback application

Definition at line 386 of file `ftm_common.h`.

### 16.56.2.1.4 `ftm_clock_source_t ftmClockSource`

Clock source used by FTM counter

Definition at line 378 of file `ftm_common.h`.

### 16.56.2.1.5 `ftm_config_mode_t ftmMode`

Mode of operation for FTM

Definition at line 379 of file `ftm_common.h`.

### 16.56.2.1.6 `uint16_t ftmModValue`

This field is used only in input capture mode to store MOD value

Definition at line 380 of file `ftm_common.h`.

### 16.56.2.1.7 `uint16_t ftmPeriod`

This field is used only in PWM mode to store signal period

Definition at line 381 of file `ftm_common.h`.

### 16.56.2.1.8 `uint32_t ftmSourceClockFrequency`

The clock frequency is used for counting

Definition at line 382 of file `ftm_common.h`.

### 16.56.2.1.9 `uint16_t measurementResults[FEATURE_FTM_CHANNEL_COUNT]`

This field is used only in input capture mode to store edges time stamps

Definition at line 383 of file `ftm_common.h`.

### 16.56.2.2 `struct ftm_pwm_sync_t`

FlexTimer Registers sync parameters Please don't use software and hardware trigger simultaneously Implements : `ftm_pwm_sync_t_Class`.

Definition at line 394 of file `ftm_common.h`.

## Data Fields

- bool [softwareSync](#)
- bool [hardwareSync0](#)

- bool [hardwareSync1](#)
- bool [hardwareSync2](#)
- bool [maxLoadingPoint](#)
- bool [minLoadingPoint](#)
- [ftm\\_reg\\_update\\_t](#) [inverterSync](#)
- [ftm\\_reg\\_update\\_t](#) [outRegSync](#)
- [ftm\\_reg\\_update\\_t](#) [maskRegSync](#)
- [ftm\\_reg\\_update\\_t](#) [initCounterSync](#)
- bool [autoClearTrigger](#)
- [ftm\\_pwm\\_sync\\_mode\\_t](#) [syncPoint](#)

## Field Documentation

### 16.56.2.2.1 bool [autoClearTrigger](#)

Available only for hardware trigger

Definition at line 412 of file [ftm\\_common.h](#).

### 16.56.2.2.2 bool [hardwareSync0](#)

True - enable hardware 0 sync, False - disable hardware 0 sync

Definition at line 398 of file [ftm\\_common.h](#).

### 16.56.2.2.3 bool [hardwareSync1](#)

True - enable hardware 1 sync, False - disable hardware 1 sync

Definition at line 400 of file [ftm\\_common.h](#).

### 16.56.2.2.4 bool [hardwareSync2](#)

True - enable hardware 2 sync, False - disable hardware 2 sync

Definition at line 402 of file [ftm\\_common.h](#).

### 16.56.2.2.5 [ftm\\_reg\\_update\\_t](#) [initCounterSync](#)

Configures CNTIN sync

Definition at line 411 of file [ftm\\_common.h](#).

### 16.56.2.2.6 [ftm\\_reg\\_update\\_t](#) [inverterSync](#)

Configures INVCTRL sync

Definition at line 408 of file [ftm\\_common.h](#).

### 16.56.2.2.7 [ftm\\_reg\\_update\\_t](#) [maskRegSync](#)

Configures OUTMASK sync

Definition at line 410 of file [ftm\\_common.h](#).

### 16.56.2.2.8 bool [maxLoadingPoint](#)

True - enable maximum loading point, False - disable maximum loading point

Definition at line 404 of file [ftm\\_common.h](#).

### 16.56.2.2.9 bool [minLoadingPoint](#)

True - enable minimum loading point, False - disable minimum loading point

Definition at line 406 of file ftm\_common.h.

#### 16.56.2.2.10 `ftm_reg_update_t` outRegSync

Configures SWOCTRL sync

Definition at line 409 of file ftm\_common.h.

#### 16.56.2.2.11 `bool` softwareSync

True - enable software sync, False - disable software sync

Definition at line 396 of file ftm\_common.h.

#### 16.56.2.2.12 `ftm_pwm_sync_mode_t` syncPoint

Configure synchronization method (waiting next loading point or immediate)

Definition at line 413 of file ftm\_common.h.

### 16.56.2.3 `struct ftm_user_config_t`

Configuration structure that the user needs to set.

Implements : `ftm_user_config_t_Class`

Definition at line 422 of file ftm\_common.h.

#### Data Fields

- [ftm\\_pwm\\_sync\\_t](#) syncMethod
- [ftm\\_config\\_mode\\_t](#) ftmMode
- [ftm\\_clock\\_ps\\_t](#) ftmPrescaler
- [ftm\\_clock\\_source\\_t](#) ftmClockSource
- [ftm\\_bdm\\_mode\\_t](#) BDMMode
- `bool` isToflsrEnabled
- `bool` [enableInitializationTrigger](#)

#### Field Documentation

##### 16.56.2.3.1 `ftm_bdm_mode_t` BDMMode

Select FTM behavior in BDM mode

Definition at line 430 of file ftm\_common.h.

##### 16.56.2.3.2 `bool` enableInitializationTrigger

true: enable the generation of initialization trigger false: disable the generation of initialization trigger

Definition at line 433 of file ftm\_common.h.

##### 16.56.2.3.3 `ftm_clock_source_t` ftmClockSource

Select clock source for FTM

Definition at line 429 of file ftm\_common.h.

##### 16.56.2.3.4 `ftm_config_mode_t` ftmMode

Mode of operation for FTM

Definition at line 426 of file ftm\_common.h.

#### 16.56.2.3.5 `ftm_clock_ps_t` `ftmPrescaler`

Register pre-scaler options available in the `ftm_clock_ps_t` enumeration

Definition at line 427 of file `ftm_common.h`.

#### 16.56.2.3.6 `bool` `isToflsrEnabled`

true: enable interrupt, false: write interrupt is disabled

Definition at line 431 of file `ftm_common.h`.

#### 16.56.2.3.7 `ftm_pwm_sync_t` `syncMethod`

Register sync options available in the `ftm_sync_method_t` enumeration

Definition at line 424 of file `ftm_common.h`.

### 16.56.3 Macro Definition Documentation

#### 16.56.3.1 `#define` `CHAN0_IDX` (0U)

Channel number for CHAN1.

Definition at line 205 of file `ftm_common.h`.

#### 16.56.3.2 `#define` `CHAN1_IDX` (1U)

Channel number for CHAN2.

Definition at line 207 of file `ftm_common.h`.

#### 16.56.3.3 `#define` `CHAN2_IDX` (2U)

Channel number for CHAN3.

Definition at line 209 of file `ftm_common.h`.

#### 16.56.3.4 `#define` `CHAN3_IDX` (3U)

Channel number for CHAN4.

Definition at line 211 of file `ftm_common.h`.

#### 16.56.3.5 `#define` `CHAN4_IDX` (4U)

Channel number for CHAN5.

Definition at line 213 of file `ftm_common.h`.

#### 16.56.3.6 `#define` `CHAN5_IDX` (5U)

Channel number for CHAN6.

Definition at line 215 of file `ftm_common.h`.

#### 16.56.3.7 `#define` `CHAN6_IDX` (6U)

Channel number for CHAN7.

Definition at line 217 of file `ftm_common.h`.

#### 16.56.3.8 `#define` `CHAN7_IDX` (7U)

Definition at line 219 of file `ftm_common.h`.

16.56.3.9 `#define FTM_RMW_CnSCV_REG( base, channel, mask, value ) (((base)->CONTROLS[channel].CnSC) = (((base)->CONTROLS[channel].CnSC) & ~(mask)) | (value))`

FTM\_CnSCV - Read and modify and write Channel (n) Status And Control (RW)

Definition at line 112 of file `ftm_common.h`.

16.56.3.10 `#define FTM_RMW_CNT( base, mask, value ) (((base)->CNT) = (((base)->CNT) & ~(mask)) | (value))`

FTM\_CNT - Read and modify and write to Counter (RW)

Definition at line 87 of file `ftm_common.h`.

16.56.3.11 `#define FTM_RMW_CNTIN( base, mask, value ) (((base)->CNTIN) = (((base)->CNTIN) & ~(mask)) | (value))`

FTM\_CNTIN - Read and modify and write Counter Initial Value (RW)

Definition at line 97 of file `ftm_common.h`.

16.56.3.12 `#define FTM_RMW_CONF( base, mask, value ) (((base)->CONF) = (((base)->CONF) & ~(mask)) | (value))`

FTM\_CONF - Read and modify and write Configuration (RW)

Definition at line 136 of file `ftm_common.h`.

16.56.3.13 `#define FTM_RMW_DEADTIME( base, mask, value ) (((base)->DEADTIME) = (((base)->DEADTIME) & ~(mask)) | (value))`

FTM\_DEADTIME - Read and modify and write Dead-time Insertion Control (RW)

Definition at line 117 of file `ftm_common.h`.

16.56.3.14 `#define FTM_RMW_EXTTRIG_REG( base, mask, value ) (((base)->EXTTRIG) = (((base)->EXTTRIG) & ~(mask)) | (value))`

FTM\_EXTTRIG - Read and modify and write External Trigger Control (RW)

Definition at line 121 of file `ftm_common.h`.

16.56.3.15 `#define FTM_RMW_FILTER( base, mask, value ) (((base)->FILTER) = (((base)->FILTER) & ~(mask)) | (value))`

FILTER - Read and modify and write Filter (RW)

Definition at line 146 of file `ftm_common.h`.

16.56.3.16 `#define FTM_RMW_FLTCTRL( base, mask, value ) (((base)->FLTCTRL) = (((base)->FLTCTRL) & ~(mask)) | (value))`

FTM\_FLTCTRL - Read and modify and write Fault Control (RW)

Definition at line 126 of file `ftm_common.h`.

16.56.3.17 `#define FTM_RMW_FMS( base, mask, value ) (((base)->FMS) = (((base)->FMS) & ~(mask)) | (value))`

FTM\_FMS - Read and modify and write Fault Mode Status (RW)

Definition at line 131 of file `ftm_common.h`.

16.56.3.18 `#define FTM_RMW_MOD( base, mask, value ) (((base)->MOD) = (((base)->MOD) & ~(mask)) | (value))`

FTM\_MOD - Read and modify and write Modulo (RW)

Definition at line 92 of file `ftm_common.h`.

16.56.3.19 `#define FTM_RMW_MODE( base, mask, value ) (((base)->MODE) = (((base)->MODE) & ~(mask)) | (value))`

FTM\_MODE - Read and modify and write Counter Features Mode Selection (RW)

Definition at line 107 of file ftm\_common.h.

16.56.3.20 `#define FTM_RMW_PAIR0DEADTIME( base, mask, value ) (((base)->PAIR0DEADTIME) = (((base)->PAIR0DEADTIME) & ~(mask)) | (value))`

FTM\_PAIR0DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 0 (RW)

Definition at line 161 of file ftm\_common.h.

16.56.3.21 `#define FTM_RMW_PAIR1DEADTIME( base, mask, value ) (((base)->PAIR1DEADTIME) = (((base)->PAIR1DEADTIME) & ~(mask)) | (value))`

FTM\_PAIR1DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 1 (RW)

Definition at line 166 of file ftm\_common.h.

16.56.3.22 `#define FTM_RMW_PAIR2DEADTIME( base, mask, value ) (((base)->PAIR2DEADTIME) = (((base)->PAIR2DEADTIME) & ~(mask)) | (value))`

FTM\_PAIR2DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 2 (RW)

Definition at line 171 of file ftm\_common.h.

16.56.3.23 `#define FTM_RMW_PAIR3DEADTIME( base, mask, value ) (((base)->PAIR3DEADTIME) = (((base)->PAIR3DEADTIME) & ~(mask)) | (value))`

FTM\_PAIR3DEADTIME - Read and modify and write Dead-time Insertion Control for the pair 3 (RW)

Channel number for CHAN0.

Definition at line 176 of file ftm\_common.h.

16.56.3.24 `#define FTM_RMW_POL( base, mask, value ) (((base)->POL) = (((base)->POL) & ~(mask)) | (value))`

POL - Read and modify and write Polarity (RW)

Definition at line 141 of file ftm\_common.h.

16.56.3.25 `#define FTM_RMW_QDCTRL( base, mask, value ) (((base)->QDCTRL) = (((base)->QDCTRL) & ~(mask)) | (value))`

QDCTRL - Read and modify and write Quadrature Decoder Control And Status (RW)

Definition at line 156 of file ftm\_common.h.

16.56.3.26 `#define FTM_RMW_SC( base, mask, value ) (((base)->SC) = (((base)->SC) & ~(mask)) | (value))`

FTM\_SC - Read and modify and write to Status And Control (RW)

Definition at line 82 of file ftm\_common.h.

16.56.3.27 `#define FTM_RMW_STATUS( base, mask, value ) (((base)->STATUS) = (((base)->STATUS) & ~(mask)) | (value))`

FTM\_STATUS - Read and modify and write Capture And Compare Status (RW)

Definition at line 102 of file ftm\_common.h.

16.56.3.28 `#define FTM_RMW_SYNC( base, mask, value ) (((base)->SYNC) = (((base)->SYNC) & ~(mask)) | (value))`

SYNC - Read and modify and write Synchronization (RW)

Definition at line 151 of file ftm\_common.h.

## 16.56.4 Enumeration Type Documentation

16.56.4.1 enum `ftm_bdm_mode_t`

Options for the FlexTimer behavior in BDM Mode.

Implements : `ftm_bdm_mode_t_Class`

## Enumerator

***FTM\_BDM\_MODE\_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

***FTM\_BDM\_MODE\_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

***FTM\_BDM\_MODE\_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

***FTM\_BDM\_MODE\_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

Definition at line 355 of file `ftm_common.h`.

16.56.4.2 enum `ftm_clock_ps_t`

FlexTimer pre-scaler factor selection for the clock source. In quadrature decoder mode set `FTM_CLOCK_DIVID_BY_1`.

Implements : `ftm_clock_ps_t_Class`

## Enumerator

***FTM\_CLOCK\_DIVID\_BY\_1*** Divide by 1

***FTM\_CLOCK\_DIVID\_BY\_2*** Divide by 2

***FTM\_CLOCK\_DIVID\_BY\_4*** Divide by 4

***FTM\_CLOCK\_DIVID\_BY\_8*** Divide by 8

***FTM\_CLOCK\_DIVID\_BY\_16*** Divide by 16

***FTM\_CLOCK\_DIVID\_BY\_32*** Divide by 32

***FTM\_CLOCK\_DIVID\_BY\_64*** Divide by 64

***FTM\_CLOCK\_DIVID\_BY\_128*** Divide by 128

Definition at line 261 of file `ftm_common.h`.

16.56.4.3 enum `ftm_clock_source_t`

FlexTimer clock source selection.

Implements : `ftm_clock_source_t_Class`

## Enumerator

***FTM\_CLOCK\_SOURCE\_NONE*** None use clock for FTM

***FTM\_CLOCK\_SOURCE\_SYSTEMCLK*** System clock

***FTM\_CLOCK\_SOURCE\_FIXEDCLK*** Fixed clock

***FTM\_CLOCK\_SOURCE\_EXTERNALCLK*** External clock

Definition at line 247 of file `ftm_common.h`.

16.56.4.4 enum `ftm_config_mode_t`

FlexTimer operation mode.

Implements : `ftm_config_mode_t_Class`

Enumerator

***FTM\_MODE\_NOT\_INITIALIZED*** The driver is not initialized  
***FTM\_MODE\_INPUT\_CAPTURE*** Input capture  
***FTM\_MODE\_OUTPUT\_COMPARE*** Output compare  
***FTM\_MODE\_EDGE\_ALIGNED\_PWM*** Edge aligned PWM  
***FTM\_MODE\_CEN\_ALIGNED\_PWM*** Center aligned PWM  
***FTM\_MODE\_QUADRATURE\_DECODER*** Quadrature decoder  
***FTM\_MODE\_UP\_TIMER*** Timer with up counter  
***FTM\_MODE\_UP\_DOWN\_TIMER*** timer with up-down counter  
***FTM\_MODE\_EDGE\_ALIGNED\_PWM\_AND\_INPUT\_CAPTURE*** Edge aligned PWM and input capture

Definition at line 229 of file `ftm_common.h`.

16.56.4.5 enum `ftm_deadtime_ps_t`

FlexTimer pre-scaler factor for the dead-time insertion.

Implements : `ftm_deadtime_ps_t_Class`

Enumerator

***FTM\_DEADTIME\_DIVID\_BY\_1*** Divide by 1  
***FTM\_DEADTIME\_DIVID\_BY\_4*** Divide by 4  
***FTM\_DEADTIME\_DIVID\_BY\_16*** Divide by 16

Definition at line 343 of file `ftm_common.h`.

16.56.4.6 enum `ftm_interrupt_option_t`

List of FTM interrupts.

Implements : `ftm_interrupt_option_t_Class`

Enumerator

***FTM\_CHANNEL0\_INT\_ENABLE*** Channel 0 interrupt  
***FTM\_CHANNEL1\_INT\_ENABLE*** Channel 1 interrupt  
***FTM\_CHANNEL2\_INT\_ENABLE*** Channel 2 interrupt  
***FTM\_CHANNEL3\_INT\_ENABLE*** Channel 3 interrupt  
***FTM\_CHANNEL4\_INT\_ENABLE*** Channel 4 interrupt  
***FTM\_CHANNEL5\_INT\_ENABLE*** Channel 5 interrupt  
***FTM\_CHANNEL6\_INT\_ENABLE*** Channel 6 interrupt  
***FTM\_CHANNEL7\_INT\_ENABLE*** Channel 7 interrupt  
***FTM\_FAULT\_INT\_ENABLE*** Fault interrupt  
***FTM\_TIME\_OVERFLOW\_INT\_ENABLE*** Time overflow interrupt  
***FTM\_RELOAD\_INT\_ENABLE*** Reload interrupt; Available only on certain SoC's

Definition at line 278 of file `ftm_common.h`.



16.56.4.7 enum `ftm_pwm_sync_mode_t`

FTM update register.

Implements : `ftm_pwm_sync_mode_t_Class`

Enumerator

***FTM\_WAIT\_LOADING\_POINTS*** FTM register is updated at first loading point

***FTM\_UPDATE\_NOW*** FTM register is updated immediately

Definition at line 332 of file `ftm_common.h`.

16.56.4.8 enum `ftm_reg_update_t`

FTM sync source.

Implements : `ftm_reg_update_t_Class`

Enumerator

***FTM\_SYSTEM\_CLOCK*** Register is updated with its buffer value at all rising edges of system clock

***FTM\_PWM\_SYNC*** Register is updated with its buffer value at the FTM synchronization

Definition at line 319 of file `ftm_common.h`.

16.56.4.9 enum `ftm_status_flag_t`

List of FTM flags.

Implements : `ftm_status_flag_t_Class`

Enumerator

***FTM\_CHANNEL0\_FLAG*** Channel 0 Flag

***FTM\_CHANNEL1\_FLAG*** Channel 1 Flag

***FTM\_CHANNEL2\_FLAG*** Channel 2 Flag

***FTM\_CHANNEL3\_FLAG*** Channel 3 Flag

***FTM\_CHANNEL4\_FLAG*** Channel 4 Flag

***FTM\_CHANNEL5\_FLAG*** Channel 5 Flag

***FTM\_CHANNEL6\_FLAG*** Channel 6 Flag

***FTM\_CHANNEL7\_FLAG*** Channel 7 Flag

***FTM\_FAULT\_FLAG*** Fault Flag

***FTM\_TIME\_OVERFLOW\_FLAG*** Time overflow Flag

***FTM\_RELOAD\_FLAG*** Reload Flag; Available only on certain SoC's

***FTM\_CHANNEL\_TRIGGER\_FLAG*** Channel trigger Flag

Definition at line 298 of file `ftm_common.h`.

## 16.56.5 Function Documentation

16.56.5.1 `static void FTM_DRV_ClearChnEventStatus ( FTM_Type *const ftmBase, uint8_t channel )` `[inline]`,  
`[static]`

Clears the FTM peripheral timer all channel event status.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number

Implements : FTM\_DRV\_ClearChnEventStatus\_Activity

Definition at line 782 of file ftm\_common.h.

**16.56.5.2** static void FTM\_DRV\_ClearChSC ( FTM\_Type \*const *ftmBase*, uint8\_t *channel* ) [inline], [static]

Clears the content of Channel (n) Status And Control.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number

Implements : FTM\_DRV\_ClearChSC\_Activity

Definition at line 529 of file ftm\_common.h.

**16.56.5.3** static void FTM\_DRV\_ClearFaultFlagDetected ( FTM\_Type \*const *ftmBase*, uint8\_t *channel* ) [inline], [static]

Clear a fault condition is detected at the fault input.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel

Implements : FTM\_DRV\_ClearFaultFlagDetected\_Activity

Definition at line 968 of file ftm\_common.h.

**16.56.5.4** void FTM\_DRV\_ClearStatusFlags ( uint32\_t *instance*, uint32\_t *flagMask* )

This function is used to clear the FTM status flags.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>flagMask</i>	The status flags to clear. This is a logical OR of members of the enumeration <a href="#">ftm_status_flag_t</a>

Definition at line 739 of file ftm\_common.c.

**16.56.5.5** uint16\_t FTM\_DRV\_ConvertFreqToPeriodTicks ( uint32\_t *instance*, uint32\_t *frequencyHz* )

This function is used to covert the given frequency to period in ticks.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>frequencyHz</i>	Frequency value in Hz.

**Returns**

The value in ticks of the frequency

Definition at line 835 of file ftm\_common.c.

**16.56.5.6** status\_t FTM\_DRV\_CounterReset ( uint32\_t *instance*, bool *softwareTrigger* )

This function will allow the FTM to restart the counter to its initial counting value in the register. Note that the configuration is set in the [FTM\\_DRV\\_SetSync\(\)](#) function to make sure that the FTM registers are updated by software trigger or hardware trigger.

## Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>softwareTrigger</i>	Selects the software trigger or hardware trigger to update COUNT register. <ul style="list-style-type: none"> <li>• true: A software trigger is generate to update register</li> <li>• false: A software trigger is not implemented and need to update later or select a hardware trigger and waiting an external trigger for updating register.</li> </ul>

Definition at line 857 of file ftm\_common.c.

#### 16.56.5.7 status\_t FTM\_DRV\_Deinit ( uint32\_t *instance* )

Shuts down the FTM driver.

## Parameters

in	<i>instance</i>	The FTM peripheral instance number.
----	-----------------	-------------------------------------

## Returns

operation status

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 196 of file ftm\_common.c.

#### 16.56.5.8 static void FTM\_DRV\_DisableFaultInt ( FTM\_Type \*const *ftmBase* ) [inline],[static]

Disables the FTM peripheral timer fault interrupt.

## Parameters

in	<i>ftmBase</i>	The FTM base address pointer
----	----------------	------------------------------

Implements : FTM\_DRV\_DisableFaultInt\_Activity

Definition at line 855 of file ftm\_common.h.

#### 16.56.5.9 void FTM\_DRV\_DisableInterrupts ( uint32\_t *instance*, uint32\_t *interruptMask* )

This function is used to disable some interrupts.

## Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>interruptMask</i>	The mask of interrupt. This is a logical OR of members of the enumeration <a href="#">ftm_interrupt_option_t</a>

Definition at line 594 of file ftm\_common.c.

#### 16.56.5.10 status\_t FTM\_DRV\_EnableInterrupts ( uint32\_t *instance*, uint32\_t *interruptMask* )

This function will enable the generation a list of interrupts. It includes the FTM overflow interrupts, the reload point interrupt, the fault interrupt and the channel (n) interrupt.

## Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>interruptMask</i>	The mask of interrupt. This is a logical OR of members of the enumeration <a href="#">ftm_interrupt_option_t</a>

**Returns**

operation status

- STATUS\_SUCCESS : Completed successfully.

Definition at line 543 of file ftm\_common.c.

**16.56.5.11 status\_t FTM\_DRV\_GenerateHardwareTrigger ( uint32\_t instance )**

This function is used to configure a trigger source for FTM instance. This allow a hardware trigger input which can be used in PWM synchronization. Note that the hardware trigger is implemented only on trigger 1 for each instance.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
----	-----------------	-------------------------------------

**Returns**

operation status

- STATUS\_SUCCESS : Completed successfully.

Definition at line 524 of file ftm\_common.c.

**16.56.5.12 static bool FTM\_DRV\_GetChInputState ( const FTM\_Type \* ftmBase, uint8\_t channel ) [inline], [static]**

Get the state of channel input.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number

**Returns**

State of the channel inputs

- true : The channel input is one
- false: The channel input is zero

Implements : FTM\_DRV\_GetChInputState\_Activity

Definition at line 695 of file ftm\_common.h.

**16.56.5.13 static uint16\_t FTM\_DRV\_GetChnCountVal ( const FTM\_Type \* ftmBase, uint8\_t channel ) [inline], [static]**

Gets the FTM peripheral timer channel counter value.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number

**Returns**

Channel counter value

Implements : FTM\_DRV\_GetChnCountVal\_Activity

Definition at line 732 of file ftm\_common.h.

```
16.56.5.14 static uint8_t FTM_DRV_GetChnEdgeLevel ( const FTM_Type * ftmBase, uint8_t channel ) [inline],
[static]
```

Gets the FTM peripheral timer channel edge level.

#### Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number

#### Returns

The ELSnB:ELSnA mode value, will be 00, 01, 10, 11

Implements : FTM\_DRV\_GetChnEdgeLevel\_Activity

Definition at line 551 of file ftm\_common.h.

```
16.56.5.15 static bool FTM_DRV_GetChnEventStatus ( const FTM_Type * ftmBase, uint8_t channel ) [inline],
[static]
```

Gets the FTM peripheral timer channel event status.

#### Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number

#### Returns

Channel event status

- true : A channel event has occurred
- false : No channel event has occurred

Implements : FTM\_DRV\_GetChnEventStatus\_Activity

Definition at line 752 of file ftm\_common.h.

```
16.56.5.16 static bool FTM_DRV_GetChOutputValue ( const FTM_Type * ftmBase, uint8_t channel ) [inline],
[static]
```

Get the value of channel output.

#### Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number

#### Returns

Value of the channel outputs

- true : The channel output is one
- false: The channel output is zero

Implements : FTM\_DRV\_GetChOutputValue\_Activity

Definition at line 714 of file ftm\_common.h.

```
16.56.5.17 static uint8_t FTM_DRV_GetClockFilterPs ( const FTM_Type * ftmBase ) [inline],[static]
```

Reads the FTM filter clock divider.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
----	----------------	------------------------------

**Returns**

The FTM filter clock pre-scale divider

Implements : FTM\_DRV\_GetClockFilterPs\_Activity

Definition at line 474 of file ftm\_common.h.

**16.56.5.18** static uint16\_t FTM\_DRV\_GetCounter ( const FTM\_Type \* *ftmBase* ) [inline], [static]

Returns the FTM peripheral current counter value.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
----	----------------	------------------------------

**Returns**

The current FTM timer counter value

Implements : FTM\_DRV\_GetCounter\_Activity

Definition at line 488 of file ftm\_common.h.

**16.56.5.19** static uint16\_t FTM\_DRV\_GetCounterInitVal ( const FTM\_Type \* *ftmBase* ) [inline], [static]

Returns the FTM peripheral counter initial value.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
----	----------------	------------------------------

**Returns**

FTM timer counter initial value

Implements : FTM\_DRV\_GetCounterInitVal\_Activity

Definition at line 516 of file ftm\_common.h.

**16.56.5.20** void FTM\_DRV\_GetDefaultConfig ( ftm\_user\_config\_t \*const *config* )

This function will get the default configuration values in the structure which is used as a common use-case.

**Parameters**

out	<i>config</i>	Pointer to the structure in which the configuration will be saved.
-----	---------------	--

**Returns**

None

Definition at line 216 of file ftm\_common.c.

**16.56.5.21** uint32\_t FTM\_DRV\_GetEnabledInterrupts ( uint32\_t *instance* )

This function will get the enabled FTM interrupts.

**Parameters**

<i>in</i>	<i>instance</i>	The FTM peripheral instance number.
-----------	-----------------	-------------------------------------

**Returns**

The enabled interrupts. This is the logical OR of members of the enumeration [ftm\\_interrupt\\_option\\_t](#)

Definition at line 643 of file `ftm_common.c`.

**16.56.5.22** `static uint32_t FTM_DRV_GetEventStatus ( const FTM_Type * ftmBase ) [inline],[static]`

Gets the FTM peripheral timer status info for all channels.

**Parameters**

<i>in</i>	<i>ftmBase</i>	The FTM base address pointer
-----------	----------------	------------------------------

**Returns**

Channel event status value

Implements : `FTM_DRV_GetEventStatus_Activity`

Definition at line 769 of file `ftm_common.h`.

**16.56.5.23** `uint32_t FTM_DRV_GetFrequency ( uint32_t instance )`

Retrieves the frequency of the clock source feeding the FTM counter.

Function will return a 0 if no clock source is selected and the FTM counter is disabled

**Parameters**

<i>in</i>	<i>instance</i>	The FTM peripheral instance number.
-----------	-----------------	-------------------------------------

**Returns**

The frequency of the clock source running the FTM counter (0 if counter is disabled)

Definition at line 791 of file `ftm_common.c`.

**16.56.5.24** `static uint16_t FTM_DRV_GetMod ( const FTM_Type * ftmBase ) [inline],[static]`

Returns the FTM peripheral counter modulo value.

**Parameters**

<i>in</i>	<i>ftmBase</i>	The FTM base address pointer
-----------	----------------	------------------------------

**Returns**

FTM timer modulo value

Implements : `FTM_DRV_GetMod_Activity`

Definition at line 502 of file `ftm_common.h`.

**16.56.5.25** `uint32_t FTM_DRV_GetStatusFlags ( uint32_t instance )`

This function will get the FTM status flags. : Regarding the duty cycle is 100% at the channel output, the match interrupt has no event due to the C(n)V and C(n+1)V value are not between CNTIN value and MOD value.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
----	-----------------	-------------------------------------

**Returns**

The status flags. This is the logical OR of members of the enumeration [ftm\\_status\\_flag\\_t](#)

Definition at line 689 of file `ftm_common.c`.

**16.56.5.26** `static bool FTM_DRV_GetTriggerControlled ( const FTM_Type * ftmBase, uint8_t channel ) [inline], [static]`

Returns whether the trigger mode is enabled.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number

**Returns**

State of the channel outputs

- true : Enabled a trigger generation on channel output
- false: PWM outputs without generating a pulse

Implements : `FTM_DRV_GetTriggerControlled_Activity`

Definition at line 676 of file `ftm_common.h`.

**16.56.5.27** `status_t FTM_DRV_Init ( uint32_t instance, const ftm_user_config_t * info, ftm_state_t * state )`

Initializes the FTM driver.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>info</i>	The FTM user configuration structure, see <a href="#">ftm_user_config_t</a> .
out	<i>state</i>	The FTM state structure of the driver.

**Returns**

operation status

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 114 of file `ftm_common.c`.

**16.56.5.28** `static bool FTM_DRV_IsChnDma ( const FTM_Type * ftmBase, uint8_t channel ) [inline], [static]`

Returns whether the FTM peripheral timer channel DMA is enabled.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number



**Returns**

State of the FTM peripheral timer channel DMA

- true : Enabled DMA transfers
- false: Disabled DMA transfers

Implements : FTM\_DRV\_IsChnDma\_Activity

Definition at line 636 of file ftm\_common.h.

**16.56.5.29** static bool FTM\_DRV\_IsChnIcrst ( const FTM\_Type \* *ftmBase*, uint8\_t *channel* ) [inline],[static]

Returns whether the FTM FTM counter is reset.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number

**Returns**

State of the FTM peripheral timer channel ICRST

- true : Enabled the FTM counter reset
- false: Disabled the FTM counter reset

Implements : FTM\_DRV\_IsChnIcrst\_Activity

Definition at line 596 of file ftm\_common.h.

**16.56.5.30** static bool FTM\_DRV\_IsFaultFlagDetected ( const FTM\_Type \* *ftmBase*, uint8\_t *channel* ) [inline],[static]

Checks whether a fault condition is detected at the fault input.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel

**Returns**

the fault condition status

- true : A fault condition was detected at the fault input
- false: No fault condition was detected at the fault input

Implements : FTM\_DRV\_IsFaultFlagDetected\_Activity

Definition at line 952 of file ftm\_common.h.

**16.56.5.31** static bool FTM\_DRV\_IsFaultInputEnabled ( const FTM\_Type \* *ftmBase* ) [inline],[static]

Checks whether the logic OR of the fault inputs is enabled.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
----	----------------	------------------------------

**Returns**

the enabled fault inputs status

- true : The logic OR of the enabled fault inputs is 1

- false: The logic OR of the enabled fault inputs is 0

Implements : FTM\_DRV\_IsFaultInputEnabled\_Activity

Definition at line 935 of file ftm\_common.h.

**16.56.5.32** static bool FTM\_DRV\_IsFtmEnable ( const FTM\_Type \* *ftmBase* ) [inline],[static]

Get status of the FTMEN bit in the FTM\_MODE register.

Parameters

in	<i>ftmBase</i>	The FTM base address pointer
----	----------------	------------------------------

Returns

the FTM Enable status

- true : TPM compatibility. Free running counter and synchronization compatible with TPM
- false: Free running counter and synchronization are different from TPM behavior

Implements : FTM\_DRV\_IsFtmEnable\_Activity

Definition at line 886 of file ftm\_common.h.

**16.56.5.33** static bool FTM\_DRV\_IsWriteProtectionEnabled ( const FTM\_Type \* *ftmBase* ) [inline],[static]

Checks whether the write protection is enabled.

Parameters

in	<i>ftmBase</i>	The FTM base address pointer
----	----------------	------------------------------

Returns

Write-protection status

- true : If enabled
- false: If not

Implements : FTM\_DRV\_IsWriteProtectionEnabled\_Activity

Definition at line 919 of file ftm\_common.h.

**16.56.5.34** status\_t FTM\_DRV\_MaskOutputChannels ( uint32\_t *instance*, uint32\_t *channelsMask*, bool *softwareTrigger* )

This function will mask the output of the channels and at match events will be ignored by the masked channels.

Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>channelsMask</i>	The mask which will select which channels will ignore match events.
in	<i>softwareTrigger</i>	If true a software trigger is generate to update PWM parameters.

Returns

success

- STATUS\_SUCCESS : Completed successfully.

Definition at line 249 of file ftm\_common.c.

**16.56.5.35** `status_t FTM_DRV_SetAllChnSoftwareOutputControl ( uint32_t instance, uint8_t channelMask, uint8_t channelValueMask, bool softwareTrigger )`

This function will control list of channels by software to force the output to specified value. Despite the odd channels are configured as HIGH/LOW, they will be inverted in the following configuration: COMP bit = 1 and CH(n)OCV and CH(n+1)OCV are HIGH. Please check software output control behavior chapter from reference manual. : When the PWM signal is configured with LOW/HIGH polarity on the channel (n). It should be set the safe state as LOW level state. However, We will have an issue with COMP bit is zero and CH(n)OCV is HIGH and CH(n+1)OCV is LOW.in the independent channel configuration. Code configuration: { .polarity = FTM\_POLARITY\_HIGH, .safeState = FTM\_POLARITY\_LOW, .enableSecondChannelOutput = true, .secondChannelPolarity = FTM\_MAIN\_DUPLICATED, }.

Workaround: Configure the safe state as HIGH level state. The expected output will be correctly controlling Should change configuration as following: { .polarity = FTM\_POLARITY\_HIGH, .safeState = FTM\_HIGH\_STATE, .enableSecondChannelOutput = true, .secondChannelPolarity = FTM\_MAIN\_DUPLICATED, }

#### Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>channelMask</i>	The mask which will configure the channels which can be software controlled.
in	<i>channelValueMask</i>	The values which will be software configured for channels.
in	<i>softwareTrigger</i>	If true a software trigger is generate to update registers.

#### Returns

success

- STATUS\_SUCCESS : Completed successfully.

Definition at line 357 of file `ftm_common.c`.

**16.56.5.36** `static void FTM_DRV_SetCaptureTestCmd ( FTM_Type *const ftmBase, bool enable ) [inline], [static]`

Enables or disables the FTM peripheral timer capture test mode.

#### Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>enable</i>	Capture Test Mode Enable <ul style="list-style-type: none"> <li>• true : Capture test mode is enabled</li> <li>• false: Capture test mode is disabled</li> </ul>

Implements : `FTM_DRV_SetCaptureTestCmd_Activity`

Definition at line 870 of file `ftm_common.h`.

**16.56.5.37** `static void FTM_DRV_SetChnDmaCmd ( FTM_Type *const ftmBase, uint8_t channel, bool enable ) [inline], [static]`

Enables or disables the FTM peripheral timer channel DMA.

#### Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number
in	<i>enable</i>	Enable DMA transfers for the channel <ul style="list-style-type: none"> <li>• true : Enabled DMA transfers</li> <li>• false: Disabled DMA transfers</li> </ul>

Implements : `FTM_DRV_SetChnDmaCmd_Activity`

Definition at line 615 of file ftm\_common.h.

**16.56.5.38** `static void FTM_DRV_SetChnIcrstCmd ( FTM_Type *const ftmBase, uint8_t channel, bool enable ) [inline], [static]`

Configure the feature of FTM counter reset by the selected input capture event.

#### Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number
in	<i>enable</i>	Enable the FTM counter reset <ul style="list-style-type: none"> <li>• true : FTM counter is reset</li> <li>• false: FTM counter is not reset</li> </ul>

Implements : FTM\_DRV\_SetChnIcrstCmd\_Activity

Definition at line 575 of file ftm\_common.h.

**16.56.5.39** `static void FTM_DRV_SetChnOutputInitStateCmd ( FTM_Type *const ftmBase, uint8_t channel, bool state ) [inline], [static]`

Sets the FTM peripheral timer channel output initial state 0 or 1.

#### Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number
in	<i>state</i>	Initial state for channels output <ul style="list-style-type: none"> <li>• true : The initialization value is 1</li> <li>• false: The initialization value is 0</li> </ul>

Implements : FTM\_DRV\_SetChnOutputInitStateCmd\_Activity

Definition at line 832 of file ftm\_common.h.

**16.56.5.40** `static void FTM_DRV_SetChnOutputMask ( FTM_Type *const ftmBase, uint8_t channel, bool mask ) [inline], [static]`

Sets the FTM peripheral timer channel output mask.

#### Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number
in	<i>mask</i>	Value to set Output Mask <ul style="list-style-type: none"> <li>• true : Channel output is masked</li> <li>• false: Channel output is not masked</li> </ul>

Implements : FTM\_DRV\_SetChnOutputMask\_Activity

Definition at line 805 of file ftm\_common.h.

**16.56.5.41** `static void FTM_DRV_SetChnSoftwareCtrlCmd ( FTM_Type *const ftmBase, uint8_t channel, bool enable ) [inline], [static]`

Enables or disables the channel software output control.

## Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	Channel to be enabled or disabled
in	<i>enable</i>	State of channel software output control <ul style="list-style-type: none"> <li>• true : To enable the channel output will be affected by software output control</li> <li>• false: To disable the channel output is unaffected</li> </ul>

Implements : FTM\_DRV\_SetChnSoftwareCtrlCmd\_Activity

Definition at line 1018 of file ftm\_common.h.

**16.56.5.42** static void FTM\_DRV\_SetChnSoftwareCtrlVal ( FTM\_Type \*const *ftmBase*, uint8\_t *channel*, bool *enable* )  
[inline], [static]

Sets the channel software output control value. Despite the odd channels are configured as HIGH/LOW, they will be inverted in the following configuration: COMP bit = 1 and CH(n)OCV and CH(n+1)OCV are HIGH. Please check Software output control behavior chapter from RM.

## Parameters

in	<i>ftmBase</i>	The FTM base address pointer.
in	<i>channel</i>	Channel to be configured
in	<i>enable</i>	State of software output control value <ul style="list-style-type: none"> <li>• true : to force 1 to the channel output</li> <li>• false: to force 0 to the channel output</li> </ul>

Implements : FTM\_DRV\_SetChnSoftwareCtrlVal\_Activity

Definition at line 1048 of file ftm\_common.h.

**16.56.5.43** static void FTM\_DRV\_SetClockFilterPs ( FTM\_Type \*const *ftmBase*, uint8\_t *filterPrescale* ) [inline],  
[static]

Sets the filter Pre-scaler divider.

## Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>filterPrescale</i>	The FTM peripheral clock pre-scale divider

Implements : FTM\_DRV\_SetClockFilterPs\_Activity

Definition at line 459 of file ftm\_common.h.

**16.56.5.44** static void FTM\_DRV\_SetCountReinitSyncCmd ( FTM\_Type \*const *ftmBase*, bool *enable* ) [inline],  
[static]

Determines if the FTM counter is re-initialized when the selected trigger for synchronization is detected.

## Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>enable</i>	FTM counter re-initialization selection <ul style="list-style-type: none"> <li>• true : To update FTM counter when triggered</li> <li>• false: To count normally</li> </ul>

Implements : FTM\_DRV\_SetCountReinitSyncCmd\_Activity

Definition at line 902 of file `ftm_common.h`.

**16.56.5.45** `static void FTM_DRV_SetDualChnInvertCmd ( FTM_Type *const ftmBase, uint8_t chnlPairNum, bool enable )`  
`[inline], [static]`

Enables or disables the channel invert for a channel pair.

#### Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>chnlPairNum</i>	The FTM peripheral channel pair number
in	<i>enable</i>	State of channel invert for a channel pair <ul style="list-style-type: none"> <li>• true : To enable channel inverting</li> <li>• false: To disable channel inversion</li> </ul>

Implements : `FTM_DRV_SetDualChnInvertCmd_Activity`

Definition at line 991 of file `ftm_common.h`.

**16.56.5.46** `static void FTM_DRV_SetExtPairDeadtimeValue ( FTM_Type *const ftmBase, uint8_t channelPair, uint8_t value )`  
`[inline], [static]`

Sets the FTM extended dead-time value for the channel pair.

#### Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channelPair</i>	The FTM peripheral channel pair (n)
in	<i>value</i>	The FTM peripheral extend pre-scale divider using the concatenation with the dead-time value

Implements : `FTM_DRV_SetExtPairDeadtimeValue_Activity`

Definition at line 1245 of file `ftm_common.h`.

**16.56.5.47** `static void FTM_DRV_SetGlobalLoadCmd ( FTM_Type *const ftmBase )` `[inline], [static]`

Set the global load mechanism.

#### Parameters

in	<i>ftmBase</i>	The FTM base address pointer
----	----------------	------------------------------

Implements : `FTM_DRV_SetGlobalLoadCmd_Activity`

Definition at line 1072 of file `ftm_common.h`.

**16.56.5.48** `static void FTM_DRV_SetGlobalTimeBaseCmd ( FTM_Type *const ftmBase, bool enable )` `[inline], [static]`

Enables or disables the FTM timer global time base.

#### Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>enable</i>	State of global time base <ul style="list-style-type: none"> <li>• true : To enable an external global time base signal</li> <li>• false: To disable an external global time base signal</li> </ul>

Implements : `FTM_DRV_SetGlobalTimeBaseCmd_Activity`

Definition at line 1216 of file `ftm_common.h`.

```
16.56.5.49 static void FTM_DRV_SetGlobalTimeBaseOutputCmd ( FTM_Type *const ftmBase, bool enable ) [inline],  
[static]
```

Enables or disables the FTM global time base signal generation to other FTM's.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>enable</i>	State of global time base signal <ul style="list-style-type: none"> <li>• true : To enable the global time base generation to other FTM instances</li> <li>• false: To disable the global time base generation to other FTM instances</li> </ul>

Implements : FTM\_DRV\_SetGlobalTimeBaseOutputCmd\_Activity

Definition at line 1200 of file ftm\_common.h.

**16.56.5.50** static void FTM\_DRV\_SetHalfCycleCmd ( FTM\_Type \*const *ftmBase*, bool *enable* ) [inline],[static]

Enable the half cycle reload.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>enable</i>	State of the half cycle match as a reload opportunity <ul style="list-style-type: none"> <li>• true : Half cycle reload is enabled</li> <li>• false: Half cycle reload is disabled</li> </ul>

Implements : FTM\_DRV\_SetHalfCycleCmd\_Activity

Definition at line 1110 of file ftm\_common.h.

**16.56.5.51** status\_t FTM\_DRV\_SetHalfCycleReloadPoint ( uint32\_t *instance*, uint16\_t *reloadPoint*, bool *softwareTrigger* )

This function configure the value of the counter which will generates an reload point.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>reloadPoint</i>	Counter value which generates the reload point.
in	<i>softwareTrigger</i>	If true a software trigger is generate to update parameters.

**Returns**

success

- STATUS\_SUCCESS : Completed successfully.

Definition at line 291 of file ftm\_common.c.

**16.56.5.52** status\_t FTM\_DRV\_SetInitialCounterValue ( uint32\_t *instance*, uint16\_t *counterValue*, bool *softwareTrigger* )

This function configure the initial counter value. The counter will get this value after an overflow event.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>counterValue</i>	Initial counter value.
in	<i>softwareTrigger</i>	If true a software trigger is generate to update parameters.

**Returns**

success

- STATUS\_SUCCESS : Completed successfully.

Definition at line 270 of file ftm\_common.c.



```
16.56.5.53 static void FTM_DRV_SetInitTrigOnReloadCmd ( FTM_Type *const ftmBase, bool enable ) [inline],  
[static]
```

Enables or disables the FTM initialization trigger on Reload Point.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>enable</i>	bit controls whether an initialization trigger is generated <ul style="list-style-type: none"> <li>• true : Trigger is generated when a reload point is reached</li> <li>• false: Trigger is generated on counter wrap events</li> </ul>

Implements : FTM\_DRV\_SetInitTrigOnReloadCmd\_Activity

Definition at line 1184 of file ftm\_common.h.

**16.56.5.54** `status_t FTM_DRV_SetInvertingControl ( uint32_t instance, uint8_t channelsPairMask, bool softwareTrigger )`

This function will configure if the second channel of a pair will be inverted or not.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>channelsPairMask</i>	The mask which will configure which channel pair will invert the second channel.
in	<i>softwareTrigger</i>	If true a software trigger is generate to update registers.

**Returns**

success

- STATUS\_SUCCESS : Completed successfully.

Definition at line 383 of file ftm\_common.c.

**16.56.5.55** `static void FTM_DRV_SetLoadCmd ( FTM_Type *const ftmBase, bool enable ) [inline],[static]`

Enable the global load.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>enable</i>	State of the global load mechanism <ul style="list-style-type: none"> <li>• true : Global Load OK enabled</li> <li>• false: Global Load OK disabled</li> </ul>

Implements : FTM\_DRV\_SetLoadCmd\_Activity

Definition at line 1087 of file ftm\_common.h.

**16.56.5.56** `static void FTM_DRV_SetLoadFreq ( FTM_Type *const ftmBase, uint8_t val ) [inline],[static]`

Sets the frequency of reload points.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>val</i>	Value of the TOF bit set frequency

Implements : FTM\_DRV\_SetLoadFreq\_Activity

Definition at line 1230 of file ftm\_common.h.

**16.56.5.57** `status_t FTM_DRV_SetModuloCounterValue ( uint32_t instance, uint16_t counterValue, bool softwareTrigger )`

This function configure the maximum counter value.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>counterValue</i>	Maximum counter value
in	<i>softwareTrigger</i>	If true a software trigger is generate to update parameters.

**Returns**

success

- STATUS\_SUCCESS : Completed successfully.

Definition at line 403 of file ftm\_common.c.

**16.56.5.58** `status_t FTM_DRV_SetOutputlevel ( uint32_t instance, uint8_t channel, uint8_t level )`

This function will set the channel edge or level on the selection of the channel mode.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>channel</i>	The channel number.
in	<i>level</i>	The level or edge selection for channel mode.

**Returns**

success

- STATUS\_SUCCESS : Completed successfully.

Definition at line 424 of file ftm\_common.c.

**16.56.5.59** `static void FTM_DRV_SetPairDeadtimeCount ( FTM_Type *const ftmBase, uint8_t channelPair, uint8_t count )`  
`[inline], [static]`

Sets the FTM dead-time value for the channel pair.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channelPair</i>	The FTM peripheral channel pair (n)
in	<i>count</i>	The FTM peripheral selects the dead-time value <ul style="list-style-type: none"> <li>• 0U : no counts inserted</li> <li>• 1U : 1 count is inserted</li> <li>• 2U : 2 count is inserted</li> <li>• ... up to a possible 63 counts</li> </ul>

Implements : FTM\_DRV\_SetPairDeadtimeCount\_Activity

Definition at line 1323 of file ftm\_common.h.

**16.56.5.60** `static void FTM_DRV_SetPairDeadtimePrescale ( FTM_Type *const ftmBase, uint8_t channelPair,`  
`ftm_deadtime_ps_t divider ) [inline], [static]`

Sets the FTM dead time divider for the channel pair.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channelPair</i>	The FTM peripheral channel pair (n)
in	<i>divider</i>	The FTM peripheral pre-scaler divider <ul style="list-style-type: none"> <li>• FTM_DEADTIME_DIVID_BY_1 : Divide by 1</li> <li>• FTM_DEADTIME_DIVID_BY_4 : Divide by 4</li> <li>• FTM_DEADTIME_DIVID_BY_16: Divide by 16</li> </ul>

Implements : FTM\_DRV\_SetPairDeadtimePrescale\_Activity

Definition at line 1284 of file ftm\_common.h.

**16.56.5.61** static void FTM\_DRV\_SetPwmLoadChnSelCmd ( FTM\_Type \*const *ftmBase*, uint8\_t *channel*, bool *enable* )  
[inline],[static]

Includes or excludes the channel in the matching process.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	Channel to be configured
in	<i>enable</i>	State of channel <ul style="list-style-type: none"> <li>• true : means include the channel in the matching process</li> <li>• false: means do not include channel in the matching process</li> </ul>

Implements : FTM\_DRV\_SetPwmLoadChnSelCmd\_Activity

Definition at line 1157 of file ftm\_common.h.

**16.56.5.62** static void FTM\_DRV\_SetPwmLoadCmd ( FTM\_Type \*const *ftmBase*, bool *enable* ) [inline],[static]

Enables or disables the loading of MOD, CNTIN and CV with values of their write buffer.

**Parameters**

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>enable</i>	State of loading updated values <ul style="list-style-type: none"> <li>• true : To enable the loading of value of their buffer</li> <li>• false: To disable the loading of value of their buffer</li> </ul>

Implements : FTM\_DRV\_SetPwmLoadCmd\_Activity

Definition at line 1133 of file ftm\_common.h.

**16.56.5.63** status\_t FTM\_DRV\_SetSoftOutChnValue ( uint32\_t *instance*, uint8\_t *channelsValues*, bool *softwareTrigger* )

This function will force the output value of a channel to a specific value. Before using this function it's mandatory to mask the match events using FTM\_DRV\_MaskOutputChannels and to enable software output control using FTM\_DRV\_SetSoftwareOutputChannelControl. : When the PWM signal is configured with LOW/HIGH polarity on the channel (n). It should be set the safe state as LOW level state. However, We will have an issue with COMP bit is zero and CH(n)OCV is HIGH and CH(n+1)OCV is LOW in the independent channel configuration. Code configuration : { .polarity = FTM\_POLARITY\_HIGH, .safeState = FTM\_POLARITY\_LOW, .enableSecondChannelOutput = true, .secondChannelPolarity = FTM\_MAIN\_DUPLICATED, }.

Workaround: Configure the safe state as HIGH level state. The expected output will be correctly controlling Should change configuration as following: { .polarity = FTM\_POLARITY\_HIGH, .safeState = FTM\_HIGH\_STATE, .enableSecondChannelOutput = true, .secondChannelPolarity = FTM\_MAIN\_DUPLICATED, }

## Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>channelsValues</i>	The values which will be software configured for channels.
in	<i>softwareTrigger</i>	If true a software trigger is generate to update registers.

## Returns

success

- STATUS\_SUCCESS : Completed successfully.

Definition at line 314 of file ftm\_common.c.

**16.56.5.64** `status_t FTM_DRV_SetSoftwareOutputChannelControl ( uint32_t instance, uint8_t channelsMask, bool softwareTrigger )`

This function will configure which output channel can be software controlled. Software output control forces the following values on channels (n) and (n+1) when the COMP bit is zero and POL bit is zero. CH(n)OC|CH(n+1)OC|CH(n)OCV|CH(n+1)OCV|Channel (n) Output | Channel (n+1) Output 0 | 0 | X | X | is not modified by SWOC | is not modified by SWOC 1 | 1 | 0 | 0 | is forced to zero | is forced to zero 1 | 1 | 0 | 1 | is forced to zero | is forced to one 1 | 1 | 1 | 0 | is forced to one | is forced to zero 1 | 1 | 1 | 1 | is forced to one | is forced to one.

Software output control forces the following values on channels (n) and (n+1) when the COMP bit is one and POL bit is zero. CH(n)OC|CH(n+1)OC|CH(n)OCV|CH(n+1)OCV|Channel (n) Output | Channel (n+1) Output 0 | 0 | X | X | is not modified by SWOC | is not modified by SWOC 1 | 1 | 0 | 0 | is forced to zero | is forced to zero 1 | 1 | 0 | 1 | is forced to zero | is forced to one 1 | 1 | 1 | 0 | is forced to one | is forced to zero 1 | 1 | 1 | 1 | is forced to one | is forced to zero

## Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>channelsMask</i>	The mask which will configure the channels which can be software controlled.
in	<i>softwareTrigger</i>	If true a software trigger is generate to update registers.

## Returns

success

- STATUS\_SUCCESS : Completed successfully.

Definition at line 334 of file ftm\_common.c.

**16.56.5.65** `status_t FTM_DRV_SetSync ( uint32_t instance, const ftm_pwm_sync_t * param )`

This function configures sync mechanism for some FTM registers (MOD, CNINT, HCR, CnV, OUTMASK, INVCTRL, SWOCTRL).

## Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>param</i>	The sync configuration structure.

## Returns

operation status

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 448 of file ftm\_common.c.

16.56.5.66 `static void FTM_DRV_SetTrigModeControlCmd ( FTM_Type *const ftmBase, uint8_t channel, bool enable )`  
`[inline],[static]`

Enables or disables the trigger generation on FTM channel outputs.

## Parameters

in	<i>ftmBase</i>	The FTM base address pointer
in	<i>channel</i>	The FTM peripheral channel number
in	<i>enable</i>	Trigger mode control <ul style="list-style-type: none"> <li>• false : Enable PWM output without generating a pulse</li> <li>• true : Disable a trigger generation on channel output</li> </ul>

Implements : FTM\_DRV\_SetTrigModeControlCmd\_Activity

Definition at line 655 of file ftm\_common.h.

## 16.56.6 Variable Documentation

16.56.6.1 `ftm_state_t* ftmStatePtr[FTM_INSTANCE_COUNT]`

Pointer to runtime state structure.

Definition at line 81 of file ftm\_common.c.

16.56.6.2 `FTM_Type* const g_ftmBase[FTM_INSTANCE_COUNT]`

Table of base addresses for FTM instances.

Definition at line 68 of file ftm\_common.c.

16.56.6.3 `const IRQn_Type g_ftmFaultIrqlId[FTM_INSTANCE_COUNT]`

Definition at line 72 of file ftm\_common.c.

16.56.6.4 `const IRQn_Type g_ftmIrqlId[FTM_INSTANCE_COUNT][FEATURE_FTM_CHANNEL_COUNT]`

Interrupt vectors for the FTM peripheral.

Definition at line 71 of file ftm\_common.c.

16.56.6.5 `const IRQn_Type g_ftmOverflowIrqlId[FTM_INSTANCE_COUNT]`

Definition at line 73 of file ftm\_common.c.

16.56.6.6 `const IRQn_Type g_ftmReloadIrqlId[FTM_INSTANCE_COUNT]`

Definition at line 74 of file ftm\_common.c.

## 16.57 FlexTimer Input Capture Driver (FTM\_IC)

### 16.57.1 Detailed Description

FlexTimer Input Capture Peripheral Driver.

#### Hardware background

The FTM of the S32K1xx is based on a 16 bits counter and supports: input capture, output compare, PWM and some instances include quadrature decoder.

#### How to use FTM driver in your application

For all operation modes (without Quadrature Decoder mode) the user need to configure [ftm\\_user\\_config\\_t](#). This structure will be used for initialization (FTM\_DRV\_Init). The next functions used are specific for each operation mode.

#### Single edge input capture mode

For this mode the user needs to configure parameters such: maximum counter value, number of channels, input capture operation mode (for single edge input are used edge detect mode) and edge alignment. All this information is included in the [ftm\\_input\\_param\\_t](#) structure.

#### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_ic_driver.c
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_common.c
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_hw_access.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
{S32SDK_PATH}\platform\drivers\inc\
{S32SDK_PATH}\platform\drivers\src\ftm\
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

#### Clock Manager Interrupt Manager (Interrupt)

Example:

```
/* The state structure of instance in the input capture mode */
ftm_state_t stateInputCapture;
#define FTM_IC_INSTANCE OUL
/* Channels configuration structure for inputCapture input capture */
ftm_input_ch_param_t inputCapture_InputCaptureChannelConfig[1] =
{
    {
        OU, /* Channel Id */
        FTM_EDGE_DETECT, /* Input capture operation Mode */
        FTM_RISING_EDGE, /* Edge alignment Mode */
        FTM_NO_MEASUREMENT, /* Signal measurement operation type */
        OU, /* Filter value */
        false, /* Filter disabled */
        true, /* Continuous mode measurement */
        NULL, /* Vector of callbacks parameters for channels events */
        NULL /* Vector of callbacks for channels events */
    }
}
```



```

};
/* Input capture configuration for inputCapture */
ftm_input_param_t inputCapture_InitConfig =
{
    1U,                                     /* Number of channels */
    65535U,                                /* Maximum count value */
    inputCapture_InitConfigChannelConfig /* Channels configuration */
};
/* Timer mode configuration for inputCapture */
/* Global configuration of inputCapture */
ftm_user_config_t inputCapture_InitConfig =
{
    {
        false,                             /* Software trigger state */
        false,                             /* Hardware trigger 1 state */
        false,                             /* Hardware trigger 2 state */
        false,                             /* Hardware trigger 3 state */
        false,                             /* Maximum loading point state */
        false,                             /* Min loading point state */
        FTM_SYSTEM_CLOCK,                  /* Update mode for INVCTRL register */
        FTM_SYSTEM_CLOCK,                  /* Update mode for SWOCTRL register */
        FTM_SYSTEM_CLOCK,                  /* Update mode for OUTMASK register */
        FTM_SYSTEM_CLOCK,                  /* Update mode for CNTIN register */
        false,                             /* Auto clear trigger state for hardware trigger */
        FTM_UPDATE_NOW,                    /* Select synchronization method */
    },
    FTM_MODE_INPUT_CAPTURE,                /* Mode of operation for FTM */
    FTM_CLOCK_DIVID_BY_4,                  /* FTM clock pre-scaler */
    FTM_CLOCK_SOURCE_SYSTEMCLK,            /* FTM clock source */
    FTM_BDM_MODE_00,                      /* FTM debug mode */
    false,                                 /* Interrupt state */
    false                                  /* Initialization trigger */
};
FTM_DRV_Init(FTM_IC_INSTANCE, &inputCapture_InitConfig, &stateInputCapture);
FTM_DRV_InitInputCapture(FTM_IC_INSTANCE, &inputCapture_InitConfig);
counter = FTM_DRV_GetInputCaptureMeasurement(FTM_IC_INSTANCE, 0U);

```

FTM\_DRV\_GetInputCaptureMeasurement is now used in interrupt mode and this function is used to save time stamps in internal buffers.

#### Edge-Aligned PWM and Input Capture mode

- Support both Edge-Aligned PWM and Input Capture mode can work over the same FTM instance.
- The guideline can be found here [FlexTimer Pulse Width Modulation Driver \(FTM\\_PWM\)](#)

#### Data Structures

- struct [ftm\\_input\\_ch\\_param\\_t](#)  
*FlexTimer driver Input capture parameters for each channel. [More...](#)*
- struct [ftm\\_input\\_param\\_t](#)  
*FlexTimer driver input capture parameters. [More...](#)*

#### Enumerations

- enum [ftm\\_input\\_op\\_mode\\_t](#) { [FTM\\_EDGE\\_DETECT](#) = 0U, [FTM\\_SIGNAL\\_MEASUREMENT](#) = 1U, [FTM\\_NO\\_OPERATION](#) = 2U }  
*Selects mode operation in the input capture.*
- enum [ftm\\_signal\\_measurement\\_mode\\_t](#) { [FTM\\_NO\\_MEASUREMENT](#) = 0x00U, [FTM\\_RISING\\_EDGE\\_PERIOD\\_MEASUREMENT](#) = 0x01U, [FTM\\_FALLING\\_EDGE\\_PERIOD\\_MEASUREMENT](#) = 0x02U, [FTM\\_PERIOD\\_ON\\_MEASUREMENT](#) = 0x03U, [FTM\\_PERIOD\\_OFF\\_MEASUREMENT](#) = 0x04U }  
*FlexTimer input capture measurement type for dual edge input capture.*
- enum [ftm\\_edge\\_alignment\\_mode\\_t](#) { [FTM\\_NO\\_PIN\\_CONTROL](#) = 0x00U, [FTM\\_RISING\\_EDGE](#) = 0x01U, [FTM\\_FALLING\\_EDGE](#) = 0x02U, [FTM\\_BOTH\\_EDGES](#) = 0x03U }  
*FlexTimer input capture edge mode as rising edge or falling edge.*

- enum `ftm_ic_op_mode_t` {  
`FTM_DISABLE_OPERATION` = 0x00U, `FTM_TIMESTAMP_RISING_EDGE` = 0x01U, `FTM_TIMESTAMP_FALLING_EDGE` = 0x02U, `FTM_TIMESTAMP_BOTH_EDGES` = 0x03U,  
`FTM_MEASURE_RISING_EDGE_PERIOD` = 0x04U, `FTM_MEASURE_FALLING_EDGE_PERIOD` = 0x05U, `FTM_MEASURE_PULSE_HIGH` = 0x06U, `FTM_MEASURE_PULSE_LOW` = 0x07U }

The measurement type for input capture mode Implements : `ftm_ic_op_mode_t_Class`.

## Functions

- status\_t `FTM_DRV_InitInputCapture` (uint32\_t instance, const `ftm_input_param_t` \*param)  
 This function configures the channel in the Input Capture mode for either getting time-stamps on edge detection or on signal measurement. When the edge specified in the captureMode argument occurs on the channel and then the FTM counter is captured into the CnV register. The user have to read the CnV register separately to get this value. The filter function is disabled if the filterVal argument passed as 0. The filter feature. is available only on channels 0,1,2,3.
- status\_t `FTM_DRV_DeinitInputCapture` (uint32\_t instance, const `ftm_input_param_t` \*param)  
 Disables input capture mode and clears FTM timer configuration.
- uint16\_t `FTM_DRV_GetInputCaptureMeasurement` (uint32\_t instance, uint8\_t channel)  
 This function is used to calculate the measurement and/or time stamps values which are read from the C(n, n+1)V registers and stored to the static buffers.
- status\_t `FTM_DRV_StartNewSignalMeasurement` (uint32\_t instance, uint8\_t channel)  
 Starts new single-shot signal measurement of the given channel.
- status\_t `FTM_IC_DRV_SetChannelMode` (uint32\_t instance, uint8\_t channel, `ftm_ic_op_mode_t` inputMode, bool enableContinuousCapture)  
 Set mode operation for channel in the input capture mode.

## 16.57.2 Data Structure Documentation

### 16.57.2.1 struct `ftm_input_ch_param_t`

FlexTimer driver Input capture parameters for each channel.

Implements : `ftm_input_ch_param_t_Class`

Definition at line 96 of file `ftm_ic_driver.h`.

## Data Fields

- uint8\_t `hwChannelId`
- `ftm_input_op_mode_t` `inputMode`
- `ftm_edge_alignment_mode_t` `edgeAlignement`
- `ftm_signal_measurement_mode_t` `measurementType`
- uint16\_t `filterValue`
- bool `filterEn`
- bool `continuousModeEn`
- void \* `channelsCallbacksParams`
- `ic_callback_t` `channelsCallbacks`

## Field Documentation

### 16.57.2.1.1 `ic_callback_t` `channelsCallbacks`

The callback function for channels events

Definition at line 106 of file `ftm_ic_driver.h`.

**16.57.2.1.2 void\* channelsCallbacksParams**

The parameters of callback functions for channels events

Definition at line 105 of file `ftm_ic_driver.h`.

**16.57.2.1.3 bool continuousModeEn**

Continuous measurement state

Definition at line 104 of file `ftm_ic_driver.h`.

**16.57.2.1.4 ftm\_edge\_alignment\_mode\_t edgeAlignement**

Edge alignment Mode for signal measurement

Definition at line 100 of file `ftm_ic_driver.h`.

**16.57.2.1.5 bool filterEn**

Input capture filter state

Definition at line 103 of file `ftm_ic_driver.h`.

**16.57.2.1.6 uint16\_t filterValue**

Filter Value

Definition at line 102 of file `ftm_ic_driver.h`.

**16.57.2.1.7 uint8\_t hwChannelId**

Physical hardware channel ID

Definition at line 98 of file `ftm_ic_driver.h`.

**16.57.2.1.8 ftm\_input\_op\_mode\_t inputMode**

FlexTimer module mode of operation

Definition at line 99 of file `ftm_ic_driver.h`.

**16.57.2.1.9 ftm\_signal\_measurement\_mode\_t measurementType**

Measurement Mode for signal measurement

Definition at line 101 of file `ftm_ic_driver.h`.

**16.57.2.2 struct ftm\_input\_param\_t**

FlexTimer driver input capture parameters.

Implements : `ftm_input_param_t_Class`

Definition at line 114 of file `ftm_ic_driver.h`.

**Data Fields**

- `uint8_t nNumChannels`
- `uint16_t nMaxCountValue`
- `ftm_input_ch_param_t * inputChConfig`

**Field Documentation****16.57.2.2.1 ftm\_input\_ch\_param\_t\* inputChConfig**

Input capture channels configuration

Definition at line 118 of file ftm\_ic\_driver.h.

#### 16.57.2.2.2 uint16\_t nMaxCountValue

Maximum counter value. Minimum value is 0 for this mode

Definition at line 117 of file ftm\_ic\_driver.h.

#### 16.57.2.2.3 uint8\_t nNumChannels

Number of input capture channel used

Definition at line 116 of file ftm\_ic\_driver.h.

### 16.57.3 Enumeration Type Documentation

#### 16.57.3.1 enum ftm\_edge\_alignment\_mode\_t

FlexTimer input capture edge mode as rising edge or falling edge.

Implements : ftm\_edge\_alignment\_mode\_t\_Class

##### Enumerator

**FTM\_NO\_PIN\_CONTROL** No trigger  
**FTM\_RISING\_EDGE** Rising edge trigger  
**FTM\_FALLING\_EDGE** Falling edge trigger  
**FTM\_BOTH\_EDGES** Rising and falling edge trigger

Definition at line 67 of file ftm\_ic\_driver.h.

#### 16.57.3.2 enum ftm\_ic\_op\_mode\_t

The measurement type for input capture mode Implements : ftm\_ic\_op\_mode\_t\_Class.

##### Enumerator

**FTM\_DISABLE\_OPERATION** Have no operation  
**FTM\_TIMESTAMP\_RISING\_EDGE** Rising edge trigger  
**FTM\_TIMESTAMP\_FALLING\_EDGE** Falling edge trigger  
**FTM\_TIMESTAMP\_BOTH\_EDGES** Rising and falling edge trigger  
**FTM\_MEASURE\_RISING\_EDGE\_PERIOD** Period measurement between two consecutive rising edges  
**FTM\_MEASURE\_FALLING\_EDGE\_PERIOD** Period measurement between two consecutive falling edges  
**FTM\_MEASURE\_PULSE\_HIGH** The time measurement taken for the pulse to remain ON or HIGH state  
**FTM\_MEASURE\_PULSE\_LOW** The time measurement taken for the pulse to remain OFF or LOW state

Definition at line 79 of file ftm\_ic\_driver.h.

#### 16.57.3.3 enum ftm\_input\_op\_mode\_t

Selects mode operation in the input capture.

Implements : ftm\_input\_op\_mode\_t\_Class

##### Enumerator

**FTM\_EDGE\_DETECT** FTM edge detect  
**FTM\_SIGNAL\_MEASUREMENT** FTM signal measurement  
**FTM\_NO\_OPERATION** FTM no operation

Definition at line 41 of file ftm\_ic\_driver.h.

16.57.3.4 enum `ftm_signal_measurement_mode_t`

FlexTimer input capture measurement type for dual edge input capture.

Implements : `ftm_signal_measurement_mode_t_Class`

## Enumerator

**FTM\_NO\_MEASUREMENT** No measurement

**FTM\_RISING\_EDGE\_PERIOD\_MEASUREMENT** Period measurement between two consecutive rising edges

**FTM\_FALLING\_EDGE\_PERIOD\_MEASUREMENT** Period measurement between two consecutive falling edges

**FTM\_PERIOD\_ON\_MEASUREMENT** The time measurement taken for the pulse to remain ON or HIGH state

**FTM\_PERIOD\_OFF\_MEASUREMENT** The time measurement taken for the pulse to remain OFF or LOW state

Definition at line 53 of file `ftm_ic_driver.h`.

## 16.57.4 Function Documentation

16.57.4.1 `status_t FTM_DRV_DeinitInputCapture ( uint32_t instance, const ftm_input_param_t * param )`

Disables input capture mode and clears FTM timer configuration.

## Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>param</i>	Configuration of the output compare channel.

## Returns

success

- `STATUS_SUCCESS` : Completed successfully.
- `STATUS_ERROR` : Error occurred.

Definition at line 333 of file `ftm_ic_driver.c`.

16.57.4.2 `uint16_t FTM_DRV_GetInputCaptureMeasurement ( uint32_t instance, uint8_t channel )`

This function is used to calculate the measurement and/or time stamps values which are read from the C(n, n+1)V registers and stored to the static buffers.

## Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>channel</i>	For getting the time stamp of the last edge (in normal input capture) this parameter represents the channel number. For getting the last measured value (in dual edge input capture) this parameter is the lowest channel number of the pair (EX: 0, 2, 4, 6).

## Returns

value The measured value

Definition at line 403 of file `ftm_ic_driver.c`.

#### 16.57.4.3 `status_t FTM_DRV_InitInputCapture ( uint32_t instance, const ftm_input_param_t * param )`

This function configures the channel in the Input Capture mode for either getting time-stamps on edge detection or on signal measurement. When the edge specified in the `captureMode` argument occurs on the channel and then the FTM counter is captured into the CnV register. The user have to read the CnV register separately to get this value. The filter function is disabled if the `filterVal` argument passed as 0. The filter feature. is available only on channels 0,1,2,3.

##### Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>param</i>	Configuration of the input capture channel.

##### Returns

###### success

- `STATUS_SUCCESS` : Completed successfully.
- `STATUS_ERROR` : Error occurred.

Definition at line 215 of file `ftm_ic_driver.c`.

#### 16.57.4.4 `status_t FTM_DRV_StartNewSignalMeasurement ( uint32_t instance, uint8_t channel )`

Starts new single-shot signal measurement of the given channel.

##### Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>channel</i>	Configuration of the output compare channel.

##### Returns

###### success

- `STATUS_SUCCESS` : Completed successfully.
- `STATUS_ERROR` : Error occurred.

Definition at line 427 of file `ftm_ic_driver.c`.

#### 16.57.4.5 `status_t FTM_IC_DRV_SetChannelMode ( uint32_t instance, uint8_t channel, ftm_ic_op_mode_t inputMode, bool enableContinuousCapture )`

Set mode operation for channel in the input capture mode.

This function will change the channel mode at run time or when stopping channel. The channel mode is selected in the `ftm_ic_op_mode_t` enumeration type.

##### Parameters

in	<i>instance</i>	The input capture instance number.
in	<i>channel</i>	The channel number.
in	<i>inputMode</i>	The channel operation mode.
in	<i>enable↔ Continuous↔ Capture</i>	Enable/disable the continuous capture mode.

##### Returns

###### success

- `STATUS_SUCCESS` : Completed successfully.

Definition at line 466 of file `ftm_ic_driver.c`.

## 16.58 FlexTimer Module Counter Driver (FTM\_MC)

### 16.58.1 Detailed Description

FlexTimer Module Counter Peripheral Driver.

#### Hardware background

The FTM of the S32K1xx is based on a 16 bits counter and supports: input capture, output compare, PWM and some instances include quadrature decoder.

#### How to use FTM driver in your application

For all operation modes (without Quadrature Decoder mode) the user need to configure [ftm\\_user\\_config\\_t](#). This structure will be used for initialization (FTM\_DRV\_Init). The next functions used are specific for each operation mode.

#### Counter mode

For this mode the user needs to configure parameters like: counter mode (up-counting or up-down counting), maximum counter value, initial counter value. All this information is included in the [ftm\\_timer\\_param\\_t](#) structure.

#### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_mc_driver.c
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_common.c
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_hw_access.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
{S32SDK_PATH}\platform\drivers\inc\
{S32SDK_PATH}\platform\drivers\src\ftm\
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

#### [Clock Manager Interrupt Manager \(Interrupt\)](#)

Example:

```
/* The state structure of instance in the input capture mode */
ftm_state_t stateTimer;
#define FTM_TIMER_INSTANCE 1UL
/* Timer mode configuration for Timer */
ftm_timer_param_t Timer_TimerConfig =
{
    FTM_MODE_UP_TIMER,          /* Counter mode */
    0U,                         /* Initial counter value */
    0x8000U                     /* Final counter value */
};

/* Global configuration of Timer*/
ftm_user_config_t Timer_InitConfig =
{
    {
        false,                 /* Software trigger state */
        false,                 /* Hardware trigger 1 state */
        false,                 /* Hardware trigger 2 state */
        false,                 /* Hardware trigger 3 state */
    }
}
```

```

        false,                /* Maximum loading point state */
        false,                /* Min loading point state */
        FTM_SYSTEM_CLOCK,    /* Update mode for INVCTRL register */
        FTM_SYSTEM_CLOCK,    /* Update mode for SWOCTRL register */
        FTM_SYSTEM_CLOCK,    /* Update mode for OUTMASK register */
        FTM_SYSTEM_CLOCK,    /* Update mode for CNTIN register */
        false,                /* Auto clear trigger state for hardware trigger */
        FTM_UPDATE_NOW,      /* Select synchronization method */
    },
    FTM_MODE_UP_TIMER,        /* Mode of operation for FTM */
    FTM_CLOCK_DIVID_BY_2,     /* FTM clock pre-scaler */
    FTM_CLOCK_SOURCE_SYSTEMCLK, /* FTM clock source */
    FTM_BDM_MODE_11,         /* FTM debug mode */
    false,                    /* Interrupt state */
    false                      /* Initialization trigger */
};
FTM_DRV_Init(FTM_TIMER_INSTANCE, &Timer_InitConfig, &stateTimer);
FTM_DRV_InitCounter(FTM_TIMER_INSTANCE, &Timer_TimerConfig);
FTM_DRV_CounterStart(FTM_TIMER_INSTANCE);

```

## Data Structures

- struct [ftm\\_timer\\_param\\_t](#)

*The configuration structure in timer mode. [More...](#)*

## Functions

- status\_t [FTM\\_DRV\\_InitCounter](#) (uint32\_t instance, const [ftm\\_timer\\_param\\_t](#) \*timer)

*Initialize the FTM counter.*

- status\_t [FTM\\_DRV\\_CounterStart](#) (uint32\_t instance)

*Starts the FTM counter.*

- status\_t [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\\_MC\\_DRV\\_GetDefaultConfig](#) ([ftm\\_timer\\_param\\_t](#) \*const config)

*This function will get the default configuration values in the structure which is used as a common use-case.*

## 16.58.2 Data Structure Documentation

### 16.58.2.1 struct [ftm\\_timer\\_param\\_t](#)

The configuration structure in timer mode.

Implements : [ftm\\_timer\\_param\\_t\\_Class](#)

Definition at line 41 of file [ftm\\_mc\\_driver.h](#).

## Data Fields

- [ftm\\_config\\_mode\\_t](#) mode
- uint16\_t [initialValue](#)
- uint16\_t [finalValue](#)

## Field Documentation

### 16.58.2.1.1 uint16\_t [finalValue](#)

Final counter value

Definition at line 45 of file [ftm\\_mc\\_driver.h](#).



## 16.58.2.1.2 uint16\_t initialValue

Initial counter value

Definition at line 44 of file ftm\_mc\_driver.h.

## 16.58.2.1.3 ftm\_config\_mode\_t mode

FTM mode

Definition at line 43 of file ftm\_mc\_driver.h.

## 16.58.3 Function Documentation

16.58.3.1 uint32\_t FTM\_DRV\_CounterRead ( uint32\_t *instance* )

Reads back the current value of the FTM counter.

## Parameters

in	<i>instance</i>	The FTM peripheral instance number.
----	-----------------	-------------------------------------

## Returns

The current counter value

Definition at line 150 of file ftm\_mc\_driver.c.

16.58.3.2 status\_t FTM\_DRV\_CounterStart ( uint32\_t *instance* )

Starts the FTM counter.

## Parameters

in	<i>instance</i>	The FTM peripheral instance number.
----	-----------------	-------------------------------------

## Returns

operation status

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 111 of file ftm\_mc\_driver.c.

16.58.3.3 status\_t FTM\_DRV\_CounterStop ( uint32\_t *instance* )

Stops the FTM counter.

## Parameters

in	<i>instance</i>	The FTM peripheral instance number.
----	-----------------	-------------------------------------

## Returns

operation status

- STATUS\_SUCCESS : Completed successfully.

Definition at line 132 of file ftm\_mc\_driver.c.

**16.58.3.4    `status_t FTM_DRV_InitCounter ( uint32_t instance, const ftm_timer_param_t * timer )`**

Initialize the FTM counter.

Starts the FTM counter. This function provides access to the FTM counter settings. The counter can be run in Up counting and Up-down counting modes. To run the counter in Free running mode, choose Up counting option and provide 0x0 value for the initialValue and 0xFFFF for finalValue. Please call this function only when FTM is used as timer/counter. User must call the FTM\_DRV\_Deinit and the FTM\_DRV\_Init to Re-Initialize the FTM before calling FTM\_DRV\_InitCounter for the second time and afterwards.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>timer</i>	Timer configuration structure.

**Returns**

operation status

- STATUS\_SUCCESS : Initialized successfully.

Definition at line 52 of file ftm\_mc\_driver.c.

**16.58.3.5    `void FTM_MC_DRV_GetDefaultConfig ( ftm_timer_param_t *const config )`**

This function will get the default configuration values in the structure which is used as a common use-case.

**Parameters**

out	<i>config</i>	Pointer to the structure in which the configuration will be saved.
-----	---------------	--

**Returns**

None

Definition at line 166 of file ftm\_mc\_driver.c.

## 16.59 FlexTimer Output Compare Driver (FTM\_OC)

### 16.59.1 Detailed Description

FlexTimer Output Compare Peripheral Driver.

#### Hardware background

The FTM of the S32K1xx is based on a 16 bits counter and supports: input capture, output compare, PWM and some instances include quadrature decoder.

#### How to use FTM driver in your application

For all operation modes (without Quadrature Decoder mode) the user need to configure `ftm_user_config_t`. This structure will be used for initialization (FTM\_DRV\_Init). The next functions used are specific for each operation mode.

#### Output compare mode

For this mode the user needs to configure maximum counter value, number of channels used and output mode for each channel (toggle/clear/set on match). This information is stored in `ftm_output_cmp_param_t` structure type and are used in FTM\_DRV\_InitOutputCompare function. Next step is to set a value for comparison with the FTM\_DRV\_UpdateOutputCompareChannel function.

#### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_oc_driver.c
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_common.c
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_hw_access.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
{S32SDK_PATH}\platform\drivers\inc\
{S32SDK_PATH}\platform\drivers\src\ftm\
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

#### Clock Manager Interrupt Manager (Interrupt)

Example:

```
/* The state structure of instance in the output compare mode */
ftm_state_t stateOutputCompare;
#define FTM_OUTPUT_COMPARE_INSTANCE 1UL
/* Channels configuration structure for PWM output compare */
ftm_output_cmp_ch_param_t PWM_OutputCompareChannelConfig[2] =
{
    {
        0U, /* Channel id */
        FTM_TOGGLE_ON_MATCH, /* Output mode */
        10000U, /* Compared value */
        false, /* External Trigger */
    },
    {
        1U, /* Channel id */
        FTM_TOGGLE_ON_MATCH, /* Output mode */
        20000U, /* Compared value */
    }
}
```

```

        false,                /* External Trigger */
    }
};

/* Output compare configuration for PWM */
ftm_output_cmp_param_t PWM_OutputCompareConfig =
{
    2U,                        /* Number of channels */
    FTM_MODE_OUTPUT_COMPARE,   /* FTM mode */
    40000U,                    /* Maximum count value */
    PWM_OutputCompareChannelConfig /* Channels configuration */
};

/* Timer mode configuration for PWM */
/* Global configuration of PWM */
ftm_user_config_t PWM_InitConfig =
{
    {
        true,                  /* Software trigger state */
        false,                 /* Hardware trigger 1 state */
        false,                 /* Hardware trigger 2 state */
        false,                 /* Hardware trigger 3 state */
        true,                  /* Maximum loading point state */
        true,                  /* Min loading point state */
        FTM_SYSTEM_CLOCK,      /* Update mode for INVCTRL register */
        FTM_SYSTEM_CLOCK,      /* Update mode for SWOCTRL register */
        FTM_SYSTEM_CLOCK,      /* Update mode for OUTMASK register */
        FTM_SYSTEM_CLOCK,      /* Update mode for CNTIN register */
        false,                 /* Auto clear trigger state for hardware trigger */
        FTM_UPDATE_NOW,        /* select synchronization method */
    },
    FTM_MODE_OUTPUT_COMPARE,    /* Mode of operation for FTM */
    FTM_CLOCK_DIVID_BY_4,       /* FTM clock pre-scaler */
    FTM_CLOCK_SOURCE_SYSTEMCLK, /* FTM clock source */
    FTM_BDM_MODE_11,           /* FTM debug mode */
    false,                     /* Interrupt state */
    false,                     /* Initialization trigger */
};

FTM_DRV_Init(FTM_OUTPUT_COMPARE_INSTANCE, &PWM_InitConfig, &stateOutputCompare);
FTM_DRV_InitOutputCompare(FTM_OUTPUT_COMPARE_INSTANCE, &PWM_OutputCompareConfig);
/* If you want to change compared value */
FTM_DRV_UpdateOutputCompareChannel(FTM_OUTPUT_COMPARE_INSTANCE, 0UL, 1500
    0U );

```

## Data Structures

- struct `ftm_output_cmp_ch_param_t`  
FlexTimer driver PWM parameters each channel in the output compare mode. [More...](#)
- struct `ftm_output_cmp_param_t`  
FlexTimer driver PWM parameters which is configured for the list of channels. [More...](#)

## Enumerations

- enum `ftm_output_compare_mode_t` { `FTM_DISABLE_OUTPUT` = 0x00U, `FTM_TOGGLE_ON_MATCH` = 0x01U, `FTM_CLEAR_ON_MATCH` = 0x02U, `FTM_SET_ON_MATCH` = 0x03U }  
FlexTimer Mode configuration for output compare mode.
- enum `ftm_output_compare_update_t` { `FTM_RELATIVE_VALUE` = 0x00U, `FTM_ABSOLUTE_VALUE` = 0x01U }  
FlexTimer input capture type of the next output compare value.

## Functions

- status\_t `FTM_DRV_InitOutputCompare` (uint32\_t instance, const `ftm_output_cmp_param_t` \*param)  
Configures the FTM to generate timed pulses (Output compare mode).
- status\_t `FTM_DRV_DeinitOutputCompare` (uint32\_t instance, const `ftm_output_cmp_param_t` \*param)  
Disables compare match output control and clears FTM timer configuration.
- status\_t `FTM_DRV_UpdateOutputCompareChannel` (uint32\_t instance, uint8\_t channel, uint16\_t nextComparematchValue, `ftm_output_compare_update_t` update, bool softwareTrigger)  
Sets the next compare match value based on the current counter value.

## 16.59.2 Data Structure Documentation

### 16.59.2.1 struct ftm\_output\_cmp\_ch\_param\_t

FlexTimer driver PWM parameters each channel in the output compare mode.

Implements : ftm\_output\_cmp\_ch\_param\_t\_Class

Definition at line 65 of file ftm\_oc\_driver.h.

#### Data Fields

- uint8\_t [hwChannelId](#)
- [ftm\\_output\\_compare\\_mode\\_t](#) chMode
- uint16\_t [comparedValue](#)
- bool [enableExternalTrigger](#)

#### Field Documentation

##### 16.59.2.1.1 ftm\_output\_compare\_mode\_t chMode

Channel output mode

Definition at line 68 of file ftm\_oc\_driver.h.

##### 16.59.2.1.2 uint16\_t comparedValue

The compared value

Definition at line 69 of file ftm\_oc\_driver.h.

##### 16.59.2.1.3 bool enableExternalTrigger

true: enable the generation of a trigger is used for on-chip modules false: disable the generation of a trigger

Definition at line 70 of file ftm\_oc\_driver.h.

##### 16.59.2.1.4 uint8\_t hwChannelId

Physical hardware channel ID

Definition at line 67 of file ftm\_oc\_driver.h.

### 16.59.2.2 struct ftm\_output\_cmp\_param\_t

FlexTimer driver PWM parameters which is configured for the list of channels.

Implements : ftm\_output\_cmp\_param\_t\_Class

Definition at line 79 of file ftm\_oc\_driver.h.

#### Data Fields

- uint8\_t [nNumOutputChannels](#)
- [ftm\\_config\\_mode\\_t](#) mode
- uint16\_t [maxCountValue](#)
- [ftm\\_output\\_cmp\\_ch\\_param\\_t](#) \* [outputChannelConfig](#)

#### Field Documentation

##### 16.59.2.2.1 uint16\_t maxCountValue

Maximum count value in ticks

Definition at line 83 of file ftm\_oc\_driver.h.

#### 16.59.2.2.2 `ftm_config_mode_t` mode

FlexTimer PWM operation mode

Definition at line 82 of file `ftm_oc_driver.h`.

#### 16.59.2.2.3 `uint8_t` `nNumOutputChannels`

Number of output compare channels

Definition at line 81 of file `ftm_oc_driver.h`.

#### 16.59.2.2.4 `ftm_output_cmp_ch_param_t*` `outputChannelConfig`

Output compare channels configuration

Definition at line 84 of file `ftm_oc_driver.h`.

### 16.59.3 Enumeration Type Documentation

#### 16.59.3.1 `enum` `ftm_output_compare_mode_t`

FlexTimer Mode configuration for output compare mode.

Implements : `ftm_output_compare_mode_t_Class`

Enumerator

**`FTM_DISABLE_OUTPUT`** No action on output pin

**`FTM_TOGGLE_ON_MATCH`** Toggle on match

**`FTM_CLEAR_ON_MATCH`** Clear on match

**`FTM_SET_ON_MATCH`** Set on match

Definition at line 41 of file `ftm_oc_driver.h`.

#### 16.59.3.2 `enum` `ftm_output_compare_update_t`

FlexTimer input capture type of the next output compare value.

Implements : `ftm_output_compare_update_t_Class`

Enumerator

**`FTM_RELATIVE_VALUE`** Next compared value is relative to current value

**`FTM_ABSOLUTE_VALUE`** Next compared value is absolute

Definition at line 54 of file `ftm_oc_driver.h`.

### 16.59.4 Function Documentation

#### 16.59.4.1 `status_t` `FTM_DRV_DeinitOutputCompare` ( `uint32_t` *instance*, `const` `ftm_output_cmp_param_t*` *param* )

Disables compare match output control and clears FTM timer configuration.

Parameters

<code>in</code>	<i>instance</i>	The FTM peripheral instance number.
-----------------	-----------------	-------------------------------------

in	<i>param</i>	Configuration of the output compare channel
----	--------------	---

**Returns****success**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 107 of file ftm\_oc\_driver.c.

**16.59.4.2** `status_t FTM_DRV_InitOutputCompare ( uint32_t instance, const ftm_output_cmp_param_t * param )`

Configures the FTM to generate timed pulses (Output compare mode).

When the FTM counter matches the value of CnV, the channel output is changed based on what is specified in the mode argument. The signal period can be modified using param->maxCountValue. After this function when the max counter value and CnV are equal. FTM\_DRV\_UpdateOutputCompareChannel function can be used to change CnV value.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>param</i>	configuration of the output compare channels

**Returns****success**

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 42 of file ftm\_oc\_driver.c.

**16.59.4.3** `status_t FTM_DRV_UpdateOutputCompareChannel ( uint32_t instance, uint8_t channel, uint16_t nextComparematchValue, ftm_output_compare_update_t update, bool softwareTrigger )`

Sets the next compare match value based on the current counter value.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>channel</i>	Configuration of the output compare channel
in	<i>nextComparematchValue</i>	Timer value in ticks until the next compare match event should appear
in	<i>update</i>	<ul style="list-style-type: none"> <li>• FTM_RELATIVE_VALUE : nextComparematchValue will be added to current counter value</li> <li>• FTM_ABSOLUTE_VALUE : nextComparematchValue will be written in counter register as it is</li> </ul>

in	<i>softwareTrigger</i>	This parameter will be true if software trigger sync is enabled and the user want to generate a software trigger (the value from buffer will be moved to register immediate or at next loading point depending on the sync configuration). Otherwise this parameter must be false and the next compared value will be stored in buffer until a trigger signal will be received.
----	------------------------	---

**Returns**

## success

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 150 of file ftm\_oc\_driver.c.



## 16.60 FlexTimer Pulse Width Modulation Driver (FTM\_PWM)

### 16.60.1 Detailed Description

FlexTimer Pulse Width Modulation Peripheral Driver.

#### Hardware background

The FlexTimer module is based on a 16 bits counter and supports: input capture, output compare, PWM and some instances include quadrature decoder.

#### How to use FTM driver in your application

For all operation modes (without Quadrature Decoder mode) the user need to configure [ftm\\_user\\_config\\_t](#). This structure will be used for initialization (FTM\_DRV\_Init). The next functions used are specific for each operation mode.

#### PWM mode

For this mode, the user needs to configure parameters such: number of PWM channels, frequency, dead time, fault channels and duty cycle, alignment (edge or center). All this information is included in the [ftm\\_pwm\\_param\\_t](#) structure.

FTM\_DRV\_UpdatePwmChannel can be used to update duty cycles at run time. If the type of update in the duty cycle when the duty cycle can have value between 0x0 (0%) and 0x8000 (100%). If the type of update in ticks when the firstEdge and secondEdge variables can have value between 0 and ftmPeriod which is stored in the state structure.

#### Safe state and polarity of the PWM channels

These 2 parameters are dependent from FTM hardware perspective, but the FTM\_PWM driver can handle them independently, so polarity field configures the final polarity measured at MCU pins and safeState field is the value of the PWM channel when fault is detected and fault is configured to use safe state, not tri-state mode. The same behavior is available for combined mode, polarity and safe state can be configured independently.

#### API code changes

In BETA 2.9.0 the following changes were made in FTM\_PWM API:

- from [ftm\\_independent\\_ch\\_param\\_t](#) the following fields were removed: levelSelect
- in [ftm\\_independent\\_ch\\_param\\_t](#) safeState field was added
- from [ftm\\_combined\\_ch\\_param\\_t](#) the following fields were removed: levelSelect, levelSelectOnNextChn.
- in [ftm\\_combined\\_ch\\_param\\_t](#) mainChannelSafeState and secondChannelSafeState fields were added.
- [ftm\\_safe\\_state\\_polarity\\_t](#) enum was changed

From application perspective the impact is the following:

- second channel can't be used in independent channels, just in combined channels
- polarity and safe state should be checked with the new API.

The advantages of the FTM\_PWM API from BETA 2.9,0 are:

- safe state and polarity can be configured without strong understatement of the Reference Manual
- dead time will be always inserted Note that:

- In the `ftm_independent_ch_param_t` structure has the "ftm\_safe\_state\_polarity\_t safeState" variable which user can configure the polarity of PWM signal on the channel n+1. and the "ftm\_second\_channel\_polarity\_t secondChannelPolarity" is only used to configure the channel n+1 in the complementary mode for the inverted or duplicated channel n.
- In the combined channel, the configuration structure has same feature as in the independent mode. It is only difference which the "ftm\_safe\_state\_polarity\_t secondChannelSafeState" variable should be set the channel (n+1) polarity if needed.
- When the fault input is disabled in the configuration the user should be set the `ftm_safe_state_polarity_t` as `FTM_LOW_STATE` for both cases in the independent and combined channel because it is not necessary.
- On S32K1xx, the clock source for Deadtime is always the System Clock, regardless of the current Clock Source used for FTM (Fixed, External).

### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_pwm_driver.c
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_common.c
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_hw_access.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
{S32SDK_PATH}\platform\drivers\inc\
{S32SDK_PATH}\platform\drivers\src\ftm\
```

#### Preprocessor symbols

No special symbols are required for this component

#### Dependencies

### Clock Manager Interrupt Manager (Interrupt)

Example:

```
/* The state structure of instance in the PWM mode */
ftm_state_t statePwm;
#define FTM_PWM_INSTANCE 1UL
/* Fault configuration structure */
ftm_pwm_fault_param_t PWM_FaultConfig =
{
    false,
    true,
    5U,
    FTM_FAULT_CONTROL_MAN_EVEN,
    {
        {
            true,
            false,
            FTM_POLARITY_HIGH,
        },
        {
            false,
            false,
            FTM_POLARITY_LOW
        },
        {
            false,
            false,
            FTM_POLARITY_LOW
        },
        {
            false,
            false,
        }
    },
    {
        false,
        false,
    },
    {
        false,
        false,
    }
};
```

```

        FTM_POLARITY_LOW          /* Channel output state on fault */
    }
}
};

/* Independent channels configuration structure for PWM */
ftm_independent_ch_param_t PWM_IndependentChannelsConfig[1] =
{
    {
        0U,                      /* hwChannelId */
        FTM_POLARITY_LOW,        /* Polarity of the PWM signal */
        4096U,                   /* Duty cycle percent 0-0x8000 */
        false,                   /* External Trigger */
        FTM_LOW_STATE,           /* Safe state of the PWM channel when faults are detected */
    }
};

/* PWM configuration for PWM */
ftm_pwm_param_t PWM_PwmConfig =
{
    1U,                          /* Number of independent PWM channels */
    0U,                          /* Number of combined PWM channels */
    FTM_MODE_EDGE_ALIGNED_PWM,   /* PWM mode */
    0U,                          /* DeadTime Value */
    FTM_DEADTIME_DIVID_BY_4,     /* DeadTime clock divider */
    7481U,                       /* PWM frequency */
    PWM_IndependentChannelsConfig, /* Independent PWM channels configuration structure */
    NULL,                        /* Combined PWM channels configuration structure */
    &PWM_FaultConfig             /* PWM fault configuration structure */
};

/* Timer mode configuration for PWM */
/* Global configuration of PWM */
ftm_user_config_t PWM_InitConfig =
{
    {
        true,                    /* Software trigger state */
        false,                   /* Hardware trigger 1 state */
        false,                   /* Hardware trigger 2 state */
        false,                   /* Hardware trigger 3 state */
        true,                    /* Maximum loading point state */
        true,                    /* Min loading point state */
        FTM_SYSTEM_CLOCK,        /* Update mode for INVCTRL register */
        FTM_SYSTEM_CLOCK,        /* Update mode for SWOCTRL register */
        FTM_SYSTEM_CLOCK,        /* Update mode for OUTMASK register */
        FTM_SYSTEM_CLOCK,        /* Update mode for CNTIN register */
        false,                   /* Auto clear trigger state for hardware trigger */
        FTM_UPDATE_NOW,          /* Select synchronization method */
    },
    FTM_MODE_EDGE_ALIGNED_PWM,   /* PWM mode */
    FTM_CLOCK_DIVID_BY_4,        /* FTM clock pre-scaler */
    FTM_CLOCK_SOURCE_SYSTEMCLK,  /* FTM clock source */
    FTM_BDM_MODE_11,            /* FTM debug mode */
    false,                      /* Interrupt state */
    false                        /* Initialization trigger */
};
FTM_DRV_Init(FTM_PWM_INSTANCE, &PWM_InitConfig, &statePwm);
FTM_DRV_InitPwm(FTM_PWM_INSTANCE, &PWM_PwmConfig);
/* It's recommended to use softwareTrigger = true */
/* The SECOND_EDGE value is used only when PWM is used in combined mode */
FTM_DRV_UpdatePwmChannel(FTM_PWM_INSTANCE, 0UL,
    FTM_PWM_UPDATE_IN_DUTY_CYCLE, 0x800U, 0x2000U, true);

```

### PWM in Modified Combine mode

For this mode the user needs to configure parameters such: number of PWM channels, frequency, dead time, fault channels and duty cycle, alignment (edge or center). All this information is included in `ftm_pwm_param_t` data type. The Modified Combine PWM mode is intended to support the generation of PWM signals where the period is not modified while the signal is being generated, but the duty cycle will be varied. `FTM_DRV_UpdatePwmChannel` can be used to update duty cycles at run time. If the type of update in the duty cycle when the duty cycle can have value between 0x0 (0%) and 0x8000 (100%). If the type of update in ticks when the firstEdge and secondEdge variables can have value between 0 and `ftmPeriod` which is stored in the state structure. In this mode, an even channel (n) and adjacent odd channel (n+1) are combined to generate a PWM signal in the channel (n) output. Thus, the channel (n) match edge is fixed and the channel (n+1) match edge can be varied.

Example:

```

/* The state structure of instance in the PWM mode */
ftm_state_t statePwm;
#define FTM_PWM_INSTANCE          0UL

```

```

/* Fault configuration structure */
ftm_pwm_fault_param_t PWM_FaultConfig =
{
    false,
    true,
    5U,
    FTM_FAULT_CONTROL_MAN_EVEN,
    {
        {
            true,
            false,
            FTM_POLARITY_HIGH,
        },
        {
            false,
            false,
            FTM_POLARITY_LOW
        },
        {
            false,
            false,
            FTM_POLARITY_LOW
        },
        {
            false,
            false,
            FTM_POLARITY_LOW
        }
    }
};

/* Combine channels configuration structure for PWM */
ftm_combined_ch_param_t flexTimer1_CombinedChannelsConfig[1] =
{
    {
        2U,
        0U,
        0U,
        true,
        true,
        FTM_POLARITY_HIGH,
        true,
        FTM_MAIN_INVERTED,
        false,
        false,
        FTM_LOW_STATE,
        FTM_LOW_STATE,
    }
};

/* PWM configuration for PWM */
ftm_pwm_param_t PWM_PwmConfig =
{
    0U,
    1U,
    FTM_MODE_EDGE_ALIGNED_PWM,
    0U,
    FTM_DEADTIME_DIVID_BY_4,
    7481U,
    NULL,
    flexTimer1_CombinedChannelsConfig,
    &PWM_FaultConfig
};

/* Timer mode configuration for PWM */
/* Global configuration of PWM */
ftm_user_config_t PWM_InitConfig =
{
    {
        true,
        false,
        false,
        false,
        true,
        true,
        FTM_SYSTEM_CLOCK,
        FTM_SYSTEM_CLOCK,
        FTM_SYSTEM_CLOCK,
        FTM_SYSTEM_CLOCK,
        false,
        FTM_UPDATE_NOW,
    },
    FTM_MODE_EDGE_ALIGNED_PWM,
    FTM_CLOCK_DIVID_BY_4,
    FTM_CLOCK_SOURCE_SYSTEMCLK,
    FTM_BDM_MODE_11,
    false,
    false
};

```

```
FTM_DRV_Init(FTM_PWM_INSTANCE, &PWM_InitConfig, &statePwm);
FTM_DRV_InitPwm(FTM_PWM_INSTANCE, &PWM_PwmConfig);
/* It's recommended to use softwareTrigger = true */
/* Only second edge can be updated when FTM is running. */
FTM_DRV_UpdatePwmChannel(FTM_PWM_INSTANCE, 0UL,
    FTM_PWM_UPDATE_IN_DUTY_CYCLE, 0x0U, 0x2000U, true);
```

### Edge-Aligned PWM and Input Capture mode

Support an additional Input Capture mode on other channels in the same FTM instance:

- The measurement range of Input Capture will depend on PWM configuration. The frequency of measured signal must be greater than frequency of PWM signal.
- For this mode, the recommended synchronization point is the next loading point (not immediately) to avoid breaking the current measurement signal from Input Capture.
- The S32CT configuration is not possible to add both drivers on the same instance, even if the driver supports this. The initialization sequences below:

```
/* The state structure of instance in the PWM and Input Capture mode */
ftm_state_t statePwmIc;
#define FTM_PWM_IC_INSTANCE 0UL
/* Fault configuration structure */
ftm_pwm_fault_param_t PWM_FaultConfig =
{
    false,
    true,
    5U,
    FTM_FAULT_CONTROL_MAN_EVEN,
    {
        {
            true,
            false,
            FTM_POLARITY_HIGH,
        },
        {
            false,
            false,
            FTM_POLARITY_LOW,
        },
        {
            false,
            false,
            FTM_POLARITY_LOW,
        },
        {
            false,
            false,
            FTM_POLARITY_LOW,
        }
    }
};
/* Combine channels configuration structure for PWM */
ftm_combined_ch_param_t flexTimer1_CombinedChannelsConfig[1] =
{
    {
        0U,
        0U,
        0x2000U,
        true,
        true,
        FTM_POLARITY_HIGH,
        true,
        FTM_MAIN_INVERTED,
        false,
        false,
        FTM_LOW_STATE,
        FTM_LOW_STATE,
    }
};
/* PWM configuration for PWM */
ftm_pwm_param_t PWM_PwmConfig =
{
    0U,
    1U,
    FTM_MODE_EDGE_ALIGNED_PWM,
    0U,
    FTM_DEADTIME_DIVID_BY_4,
    7481U,
    /* Number of independent PWM channels */
    /* Number of combined PWM channels */
    /* PWM mode */
    /* DeadTime Value */
    /* DeadTime clock divider */
    /* PWM frequency */
};
```

```

    NULL,                                /* Independent PWM channels configuration structure */
    flexTimer1_CombinedChannelsConfig,    /* Combined PWM channels configuration structure */
    &PWM_FaultConfig                      /* PWM fault configuration structure */
};
/* Channels configuration structure for inputCapture input capture */
ftm_input_ch_param_t inputCapture_InputCaptureChannelConfig[1] =
{
    {
        3U,                                /* Channel 3 (Make sure that the channel ID must different than PWM
        channels) */
        FTM_EDGE_DETECT,                   /* Input capture operation Mode */
        FTM_RISING_EDGE,                   /* Edge alignment Mode */
        FTM_NO_MEASUREMENT,                /* Signal measurement operation type */
        0U,                                /* Filter value */
        false,                             /* Filter disabled */
        true,                              /* Continuous mode measurement */
        NULL,                              /* Vector of callbacks parameters for channels events */
        NULL                               /* Vector of callbacks for channels events */
    }
};
/* Input capture configuration for inputCapture */
ftm_input_param_t inputCapture_InputCaptureConfig =
{
    1U,                                    /* Number of channels */
    65535U,                                /* Maximum count value (This value is ignored and replaced by
    period value calculated from PWM configuration) */
    inputCapture_InputCaptureChannelConfig /* Channels configuration */
};
/* Timer mode configuration for PWM and Input Capture */
ftm_user_config_t PWM_IC_InitConfig =
{
    {
        true,                             /* Software trigger state */
        false,                             /* Hardware trigger 1 state */
        false,                             /* Hardware trigger 2 state */
        false,                             /* Hardware trigger 3 state */
        true,                              /* Maximum loading point state */
        true,                              /* Min loading point state */
        FTM_SYSTEM_CLOCK,                  /* Update mode for INVCTRL register */
        FTM_SYSTEM_CLOCK,                  /* Update mode for SWOCTRL register */
        FTM_SYSTEM_CLOCK,                  /* Update mode for OUTMASK register */
        FTM_SYSTEM_CLOCK,                  /* Update mode for CNTIN register */
        false,                             /* Auto clear trigger state for hardware trigger */
        FTM_WAIT_LOADING_POINTS,           /* Select synchronization method (Next Loading
        Point is recommended) */
    },
    FTM_MODE_EDGE_ALIGNED_PWM,             /* Edge-Align PWM mode that can be used with
    Input Capture mode */
    FTM_CLOCK_DIVID_BY_4,                  /* FTM clock pre-scaler */
    FTM_CLOCK_SOURCE_SYSTEMCLK,            /* FTM clock source */
    FTM_BDM_MODE_11,                      /* FTM debug mode */
    false,                                 /* Interrupt state */
    false                                  /* Initialization trigger */
};

/* Initialize FTM module */
FTM_DRV_Init(FTM_PWM_IC_INSTANCE, &PWM_IC_InitConfig, &statePwmIc);
/* Initialize PWM configuration (Clock configuration for both PWM and Input Capture mode)
 * Initialize the additional Input Capture mode for other channels
 * FTM_DRV_InitPwm() must be called first before user calls FTM_DRV_InitInputCapture()
 */
FTM_DRV_InitPwm(FTM_PWM_IC_INSTANCE, &PWM_PwmConfig);
FTM_DRV_InitInputCapture(FTM_PWM_IC_INSTANCE, &inputCapture_InputCaptureConfig);

...

/* De-initialize Input Capture first
 * PWM signal still work normally
 */
FTM_DRV_DeinitInputCapture(FTM_PWM_IC_INSTANCE, &inputCapture_InputCaptureConfig);
/* De-initialize PWM (FTM counter will be disabled) */
FTM_DRV_DeinitPwm(FTM_PWM_IC_INSTANCE);

```

## Data Structures

- struct `ftm_pwm_ch_fault_param_t`  
FlexTimer driver PWM Fault channel parameters. [More...](#)
- struct `ftm_pwm_fault_param_t`  
FlexTimer driver PWM Fault parameter. [More...](#)
- struct `ftm_independent_ch_param_t`  
FlexTimer driver independent PWM parameter. [More...](#)

- struct [ftm\\_combined\\_ch\\_param\\_t](#)  
*FlexTimer driver combined PWM parameter. [More...](#)*
- struct [ftm\\_pwm\\_param\\_t](#)  
*FlexTimer driver PWM parameters. [More...](#)*

## Macros

- #define [FTM\\_MAX\\_DUTY\\_CYCLE](#) (0x8000U)  
*Maximum value for PWM duty cycle.*
- #define [FTM\\_DUTY\\_TO\\_TICKS\\_SHIFT](#) (15U)  
*Shift value which converts duty to ticks.*

## Enumerations

- enum [ftm\\_pwm\\_update\\_option\\_t](#) { [FTM\\_PWM\\_UPDATE\\_IN\\_DUTY\\_CYCLE](#) = 0x00U, [FTM\\_PWM\\_UPDATE\\_IN\\_TICKS](#) = 0x01U }  
*FlexTimer Configure type of PWM update in the duty cycle or in ticks.*
- enum [ftm\\_polarity\\_t](#) { [FTM\\_POLARITY\\_LOW](#) = 0x00U, [FTM\\_POLARITY\\_HIGH](#) = 0x01U }  
*The polarity of the channel output is configured in PWM signal.*
- enum [ftm\\_second\\_channel\\_polarity\\_t](#) { [FTM\\_MAIN\\_INVERTED](#) = 0x01U, [FTM\\_MAIN\\_DUPLICATED](#) = 0x00U }  
*FlexTimer PWM channel (n+1) polarity for combine mode.*
- enum [ftm\\_fault\\_mode\\_t](#) { [FTM\\_FAULT\\_CONTROL\\_DISABLED](#) = 0x00U, [FTM\\_FAULT\\_CONTROL\\_MANUAL\\_EVEN](#) = 0x01U, [FTM\\_FAULT\\_CONTROL\\_MANUAL\\_ALL](#) = 0x02U, [FTM\\_FAULT\\_CONTROL\\_AUTO\\_ALL](#) = 0x03U }  
*FlexTimer fault control.*
- enum [ftm\\_safe\\_state\\_polarity\\_t](#) { [FTM\\_LOW\\_STATE](#) = 0x00U, [FTM\\_HIGH\\_STATE](#) = 0x01U }  
*Select level of the channel (n) output at the beginning.*

## Functions

- status\_t [FTM\\_DRV\\_DeinitPwm](#) (uint32\_t instance)  
*Stops all PWM channels.*
- status\_t [FTM\\_DRV\\_InitPwm](#) (uint32\_t instance, const [ftm\\_pwm\\_param\\_t](#) \*param)  
*Configures the duty cycle and frequency and starts the output of the PWM on all channels configured in the param structure. The independent channel configuration need to clarify the polarity and safe state as following:*
- status\_t [FTM\\_DRV\\_UpdatePwmChannel](#) (uint32\_t instance, uint8\_t channel, [ftm\\_pwm\\_update\\_option\\_t](#) typeOfUpdate, uint16\_t firstEdge, uint16\_t secondEdge, bool softwareTrigger)  
*This function updates the waveform output in PWM mode (duty cycle and phase).*
- status\_t [FTM\\_DRV\\_FastUpdatePwmChannels](#) (uint32\_t instance, uint8\_t numberOfChannels, const uint8\_t \*channels, const uint16\_t \*duty, bool softwareTrigger)  
*This function will update the duty cycle of PWM output for multiple channels.*
- status\_t [FTM\\_DRV\\_UpdatePwmPeriod](#) (uint32\_t instance, [ftm\\_pwm\\_update\\_option\\_t](#) typeOfUpdate, uint32\_t newValue, bool softwareTrigger)  
*This function will update the new period in the frequency or in the counter value into mode register which modify the period of PWM signal on the channel output.*
- status\_t [FTM\\_DRV\\_ControlChannelOutput](#) (uint32\_t instance, uint8\_t channel, bool enableChannelOutput)  
*This function is used to control the final logic of the channel output.*

## 16.60.2 Data Structure Documentation

### 16.60.2.1 struct ftm\_pwm\_ch\_fault\_param\_t

FlexTimer driver PWM Fault channel parameters.

Implements : ftm\_pwm\_ch\_fault\_param\_t\_Class

Definition at line 112 of file ftm\_pwm\_driver.h.

#### Data Fields

- bool [faultChannelEnabled](#)
- bool [faultFilterEnabled](#)
- [ftm\\_polarity\\_t](#) ftmFaultPinPolarity

#### Field Documentation

##### 16.60.2.1.1 bool faultChannelEnabled

Fault channel state

Definition at line 114 of file ftm\_pwm\_driver.h.

##### 16.60.2.1.2 bool faultFilterEnabled

Fault channel filter state

Definition at line 115 of file ftm\_pwm\_driver.h.

##### 16.60.2.1.3 ftm\_polarity\_t ftmFaultPinPolarity

Channel output state on fault

Definition at line 116 of file ftm\_pwm\_driver.h.

### 16.60.2.2 struct ftm\_pwm\_fault\_param\_t

FlexTimer driver PWM Fault parameter.

Implements : ftm\_pwm\_fault\_param\_t\_Class

Definition at line 124 of file ftm\_pwm\_driver.h.

#### Data Fields

- bool [pwmOutputStateOnFault](#)
- bool [pwmFaultInterrupt](#)
- uint8\_t [faultFilterValue](#)
- [ftm\\_fault\\_mode\\_t](#) faultMode
- [ftm\\_pwm\\_ch\\_fault\\_param\\_t](#) ftmFaultChannelParam [FTM\_FEATURE\_FAULT\_CHANNELS]

#### Field Documentation

##### 16.60.2.2.1 uint8\_t faultFilterValue

Fault filter value

Definition at line 128 of file ftm\_pwm\_driver.h.

##### 16.60.2.2.2 ftm\_fault\_mode\_t faultMode

Fault mode

Definition at line 129 of file ftm\_pwm\_driver.h.



**16.60.2.2.3** `ftm_pwm_ch_fault_param_t` `ftmFaultChannelParam`[FTM\_FEATURE\_FAULT\_CHANNELS]

Fault channels configuration

Definition at line 130 of file `ftm_pwm_driver.h`.

**16.60.2.2.4** `bool` `pwmFaultInterrupt`

PWM fault interrupt state

Definition at line 127 of file `ftm_pwm_driver.h`.

**16.60.2.2.5** `bool` `pwmOutputStateOnFault`

Output pin state on fault (safe state or tri-state)

Definition at line 126 of file `ftm_pwm_driver.h`.

**16.60.2.3** `struct` `ftm_independent_ch_param_t`

FlexTimer driver independent PWM parameter.

Implements : `ftm_independent_ch_param_t_Class`

Definition at line 138 of file `ftm_pwm_driver.h`.

**Data Fields**

- `uint8_t` `hwChannelId`
- `ftm_polarity_t` `polarity`
- `uint16_t` `uDutyCyclePercent`
- `bool` `enableExternalTrigger`
- `ftm_safe_state_polarity_t` `safeState`
- `bool` `enableSecondChannelOutput`
- `ftm_second_channel_polarity_t` `secondChannelPolarity`
- `bool` `deadTime`

**Field Documentation****16.60.2.3.1** `bool` `deadTime`

Enable/disable dead time for channel

Definition at line 150 of file `ftm_pwm_driver.h`.

**16.60.2.3.2** `bool` `enableExternalTrigger`

true: enable the generation of a trigger is used for on-chip modules false: disable the generation of a trigger

Definition at line 144 of file `ftm_pwm_driver.h`.

**16.60.2.3.3** `bool` `enableSecondChannelOutput`

Enable complementary mode on next channel

Definition at line 148 of file `ftm_pwm_driver.h`.

**16.60.2.3.4** `uint8_t` `hwChannelId`

Physical hardware channel ID

Definition at line 140 of file `ftm_pwm_driver.h`.

**16.60.2.3.5** `ftm_polarity_t` `polarity`

Polarity of the PWM signal generated on MCU pin.

Definition at line 141 of file `ftm_pwm_driver.h`.

#### 16.60.2.3.6 `ftm_safe_state_polarity_t` `safeState`

Logical state of the PWM channel `n` when an fault is detected and to set up the polarity of PWM signal on the channel `(n+1)`

Definition at line 146 of file `ftm_pwm_driver.h`.

#### 16.60.2.3.7 `ftm_second_channel_polarity_t` `secondChannelPolarity`

Polarity of the channel `n+1` relative to channel `n` in the complementary mode

Definition at line 149 of file `ftm_pwm_driver.h`.

#### 16.60.2.3.8 `uint16_t` `uDutyCyclePercent`

PWM pulse width, value should be between 0 (0%) to `FTM_MAX_DUTY_CYCLE` (100%)

Definition at line 142 of file `ftm_pwm_driver.h`.

#### 16.60.2.4 `struct` `ftm_combined_ch_param_t`

FlexTimer driver combined PWM parameter.

Implements : `ftm_combined_ch_param_t` Class

Definition at line 159 of file `ftm_pwm_driver.h`.

##### Data Fields

- `uint8_t` `hwChannelId`
- `uint16_t` `firstEdge`
- `uint16_t` `secondEdge`
- `bool` `deadTime`
- `bool` `enableModifiedCombine`
- `ftm_polarity_t` `mainChannelPolarity`
- `bool` `enableSecondChannelOutput`
- `ftm_second_channel_polarity_t` `secondChannelPolarity`
- `bool` `enableExternalTrigger`
- `bool` `enableExternalTriggerOnNextChn`
- `ftm_safe_state_polarity_t` `mainChannelSafeState`
- `ftm_safe_state_polarity_t` `secondChannelSafeState`

##### Field Documentation

#### 16.60.2.4.1 `bool` `deadTime`

Enable/disable dead time for channel

Definition at line 166 of file `ftm_pwm_driver.h`.

#### 16.60.2.4.2 `bool` `enableExternalTrigger`

The generation of the channel (`n`) trigger true: enable the generation of a trigger on the channel (`n`) false: disable the generation of a trigger on the channel (`n`)

Definition at line 171 of file `ftm_pwm_driver.h`.

#### 16.60.2.4.3 `bool` `enableExternalTriggerOnNextChn`

The generation of the channel (`n+1`) trigger true: enable the generation of a trigger on the channel (`n+1`) false: disable the generation of a trigger on the channel (`n+1`)

Definition at line 174 of file `ftm_pwm_driver.h`.

**16.60.2.4.4 bool enableModifiedCombine**

Enable/disable the modified combine mode for channels (n) and (n+1)

Definition at line 167 of file ftm\_pwm\_driver.h.

**16.60.2.4.5 bool enableSecondChannelOutput**

Select if channel (n+1) output is enabled/disabled for the complementary mode

Definition at line 169 of file ftm\_pwm\_driver.h.

**16.60.2.4.6 uint16\_t firstEdge**

First edge time. This time is relative to signal period. The value for this parameter is between 0 and FTM\_MAX\_DUTY\_CYCLE(0 = 0% from period and FTM\_MAX\_DUTY\_CYCLE = 100% from period)

Definition at line 162 of file ftm\_pwm\_driver.h.

**16.60.2.4.7 uint8\_t hwChannelId**

Physical hardware channel ID for channel (n)

Definition at line 161 of file ftm\_pwm\_driver.h.

**16.60.2.4.8 ftm\_polarity\_t mainChannelPolarity**

Polarity of the PWM signal generated on MCU pin for channel n.

Definition at line 168 of file ftm\_pwm\_driver.h.

**16.60.2.4.9 ftm\_safe\_state\_polarity\_t mainChannelSafeState**

The selection of the channel (n) state when fault is detected

Definition at line 177 of file ftm\_pwm\_driver.h.

**16.60.2.4.10 ftm\_second\_channel\_polarity\_t secondChannelPolarity**

Select channel (n+1) polarity relative to channel (n) in the complementary mode

Definition at line 170 of file ftm\_pwm\_driver.h.

**16.60.2.4.11 ftm\_safe\_state\_polarity\_t secondChannelSafeState**

The selection of the channel (n+1) state when fault is detected and set up the polarity of PWM signal on the channel (n+1)

Definition at line 178 of file ftm\_pwm\_driver.h.

**16.60.2.4.12 uint16\_t secondEdge**

Second edge time. This time is relative to signal period. The value for this parameter is between 0 and FTM\_MAX\_DUTY\_CYCLE(0 = 0% from period and FTM\_MAX\_DUTY\_CYCLE = 100% from period)

Definition at line 164 of file ftm\_pwm\_driver.h.

**16.60.2.5 struct ftm\_pwm\_param\_t**

FlexTimer driver PWM parameters.

Implements : ftm\_pwm\_param\_t\_Class

Definition at line 187 of file ftm\_pwm\_driver.h.

## Data Fields

- [uint8\\_t nNumIndependentPwmChannels](#)
- [uint8\\_t nNumCombinedPwmChannels](#)
- [ftm\\_config\\_mode\\_t mode](#)
- [uint8\\_t deadTimeValue](#)
- [ftm\\_deadtime\\_ps\\_t deadTimePrescaler](#)
- [uint32\\_t uFrequencyHZ](#)
- [ftm\\_independent\\_ch\\_param\\_t \\* pwmIndependentChannelConfig](#)
- [ftm\\_combined\\_ch\\_param\\_t \\* pwmCombinedChannelConfig](#)
- [ftm\\_pwm\\_fault\\_param\\_t \\* faultConfig](#)

## Field Documentation

### 16.60.2.5.1 [ftm\\_deadtime\\_ps\\_t deadTimePrescaler](#)

Dead time pre-scaler value[ticks]

Definition at line 193 of file [ftm\\_pwm\\_driver.h](#).

### 16.60.2.5.2 [uint8\\_t deadTimeValue](#)

Dead time value in [ticks]

Definition at line 192 of file [ftm\\_pwm\\_driver.h](#).

### 16.60.2.5.3 [ftm\\_pwm\\_fault\\_param\\_t\\* faultConfig](#)

Configuration for PWM fault

Definition at line 197 of file [ftm\\_pwm\\_driver.h](#).

### 16.60.2.5.4 [ftm\\_config\\_mode\\_t mode](#)

FTM mode

Definition at line 191 of file [ftm\\_pwm\\_driver.h](#).

### 16.60.2.5.5 [uint8\\_t nNumCombinedPwmChannels](#)

Number of combined PWM channels

Definition at line 190 of file [ftm\\_pwm\\_driver.h](#).

### 16.60.2.5.6 [uint8\\_t nNumIndependentPwmChannels](#)

Number of independent PWM channels

Definition at line 189 of file [ftm\\_pwm\\_driver.h](#).

### 16.60.2.5.7 [ftm\\_combined\\_ch\\_param\\_t\\* pwmCombinedChannelConfig](#)

Configuration for combined PWM channels

Definition at line 196 of file [ftm\\_pwm\\_driver.h](#).

### 16.60.2.5.8 [ftm\\_independent\\_ch\\_param\\_t\\* pwmIndependentChannelConfig](#)

Configuration for independent PWM channels

Definition at line 195 of file [ftm\\_pwm\\_driver.h](#).

### 16.60.2.5.9 [uint32\\_t uFrequencyHZ](#)

PWM period in Hz

Definition at line 194 of file ftm\_pwm\_driver.h.

### 16.60.3 Macro Definition Documentation

#### 16.60.3.1 #define FTM\_DUTY\_TO\_TICKS\_SHIFT (15U)

Shift value which converts duty to ticks.

Definition at line 42 of file ftm\_pwm\_driver.h.

#### 16.60.3.2 #define FTM\_MAX\_DUTY\_CYCLE (0x8000U)

Maximum value for PWM duty cycle.

Definition at line 40 of file ftm\_pwm\_driver.h.

### 16.60.4 Enumeration Type Documentation

#### 16.60.4.1 enum ftm\_fault\_mode\_t

FlexTimer fault control.

Implements : ftm\_fault\_mode\_t\_Class

#### Enumerator

**FTM\_FAULT\_CONTROL\_DISABLED** Fault control is disabled for all channels

**FTM\_FAULT\_CONTROL\_MAN\_EVEN** Fault control is enabled for even channels only (channels 0, 2, 4, and 6), and the selected mode is the manual fault clearing

**FTM\_FAULT\_CONTROL\_MAN\_ALL** Fault control is enabled for all channels, and the selected mode is the manual fault clearing

**FTM\_FAULT\_CONTROL\_AUTO\_ALL** Fault control is enabled for all channels, and the selected mode is the automatic fault clearing

Definition at line 84 of file ftm\_pwm\_driver.h.

#### 16.60.4.2 enum ftm\_polarity\_t

The polarity of the channel output is configured in PWM signal.

Implements : ftm\_polarity\_t\_Class

#### Enumerator

**FTM\_POLARITY\_LOW** The channel polarity is active LOW which is defined again

**FTM\_POLARITY\_HIGH** The channel polarity is active HIGH which is defined again

Definition at line 60 of file ftm\_pwm\_driver.h.

#### 16.60.4.3 enum ftm\_pwm\_update\_option\_t

FlexTimer Configure type of PWM update in the duty cycle or in ticks.

Implements : ftm\_pwm\_update\_option\_t\_Class

#### Enumerator

**FTM\_PWM\_UPDATE\_IN\_DUTY\_CYCLE** The type of PWM update in the duty cycle/pulse or also use in frequency update

**FTM\_PWM\_UPDATE\_IN\_TICKS** The type of PWM update in ticks

Definition at line 49 of file ftm\_pwm\_driver.h.

#### 16.60.4.4 enum `ftm_safe_state_polarity_t`

Select level of the channel (n) output at the beginning.

Implements : `ftm_safe_state_polarity_t_Class`

##### Enumerator

**`FTM_LOW_STATE`** When fault is detected PWM channel is low.

**`FTM_HIGH_STATE`** When fault is detected PWM channel is high.

Definition at line 101 of file `ftm_pwm_driver.h`.

#### 16.60.4.5 enum `ftm_second_channel_polarity_t`

FlexTimer PWM channel (n+1) polarity for combine mode.

Implements : `ftm_second_channel_polarity_t_Class`

##### Enumerator

**`FTM_MAIN_INVERTED`** The channel (n+1) output is the inverse of the channel (n) output

**`FTM_MAIN_DUPLICATED`** The channel (n+1) output is the same as the channel (n) output

Definition at line 71 of file `ftm_pwm_driver.h`.

### 16.60.5 Function Documentation

#### 16.60.5.1 `status_t FTM_DRV_ControlChannelOutput ( uint32_t instance, uint8_t channel, bool enableChannelOutput )`

This function is used to control the final logic of the channel output.

##### Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>channel</i>	The channel which is used in PWM mode.
in	<i>enable↔ ChannelOutput</i>	Enable/disable the channel output.

##### Returns

success

- `STATUS_SUCCESS` : Completed successfully.

Definition at line 675 of file `ftm_pwm_driver.c`.

#### 16.60.5.2 `status_t FTM_DRV_DeinitPwm ( uint32_t instance )`

Stops all PWM channels .

##### Parameters

in	<i>instance</i>	The FTM peripheral instance number.
----	-----------------	-------------------------------------

##### Returns

counter the current counter value

Definition at line 382 of file `ftm_pwm_driver.c`.

#### 16.60.5.3 `status_t FTM_DRV_FastUpdatePwmChannels ( uint32_t instance, uint8_t numberOfChannels, const uint8_t * channels, const uint16_t * duty, bool softwareTrigger )`

This function will update the duty cycle of PWM output for multiple channels.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>numberOfChannels</i>	The number of channels which should be updated.
in	<i>channels</i>	The list of channels which should be updated.
in	<i>duty</i>	The list of duty cycles for selected channels.
in	<i>softwareTrigger</i>	If true a software trigger is generate to update PWM parameters.

**Returns**

success

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 643 of file ftm\_pwm\_driver.c.

#### 16.60.5.4 status\_t FTM\_DRV\_InitPwm ( uint32\_t instance, const ftm\_pwm\_param\_t \* param )

Configures the duty cycle and frequency and starts the output of the PWM on all channels configured in the param structure. The independent channel configuration need to clarify the polarity and safe state as following:

- In the first channel, the POL bit is the value of safeState variable. In the second channel, the POL bit is the same value of safeState with the inverted channel and the POL bit is inverted safeState with the duplicated channel.
- If the polarity and safe state are the value, it will be Low-true pulses. It means the ELSB:ELSA = 0:1. Otherwise, it will be High-true pulses. It means the ELSB:ELSA = 1:0. Regarding the combined channel configuration:
- In both channels, the POL bit is the same value with the safeState variable
- If the polarity and safe state are the value, it will be Low-true pulses. It means the ELSB:ELSA = 0:1. Otherwise, it will be High-true pulses. It means the ELSB:ELSA = 1:0.
- COMP bit will be true when the polarity and safeState are the same value, the second channel is inverted .the first channel or when the polarity and safeState are difference value, the second channel is duplicated the first channel.
- COMP bit will be false when the polarity and safeState are the same value, the second channel is duplicated .the first channel or when the polarity and safeState are difference value, the second channel is inverted the first channel.

: These configuration will impact to the FTM\_DRV\_SetSoftwareOutputChannelControl and FTM\_DRV\_SetAllChannelsSoftwareOutputControl function. Because the software output control behavior depends on the polarity and COMP bit.

**Parameters**

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>param</i>	FTM driver PWM parameter to configure PWM options.

**Returns**

success

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 258 of file ftm\_pwm\_driver.c.

**16.60.5.5** `status_t FTM_DRV_UpdatePwmChannel ( uint32_t instance, uint8_t channel, ftm_pwm_update_option_t typeOfUpdate, uint16_t firstEdge, uint16_t secondEdge, bool softwareTrigger )`

This function updates the waveform output in PWM mode (duty cycle and phase).

: Regarding the type of updating PWM in the duty cycle, if the expected duty is 100% then the value that is to be written to hardware will be exceed value of period. It means that the FTM counter will not match the value of the CnV register in this case.

#### Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>channel</i>	The channel number. In combined mode, the code finds the channel.
in	<i>typeOfUpdate</i>	The type of PWM update in the duty cycle/pulse or in ticks.
in	<i>firstEdge</i>	Duty cycle or first edge time for PWM mode. Can take value between 0 - F <sub>TM</sub> _MAX_DUTY_CYCLE(0 = 0% from period and F <sub>TM</sub> _MAX_DUTY_CYCLE = 100% from period) Or value in ticks for the first of the PWM mode in which can have value between 0 and ftmPeriod is stored in the state structure.
in	<i>secondEdge</i>	Second edge time - only for combined mode. Can take value between 0 - F <sub>TM</sub> _MAX_DUTY_CYCLE(0 = 0% from period and F <sub>TM</sub> _MAX_DUTY_CYCLE = 100% from period). Or value in ticks for the second of the PWM mode in which can have value between 0 and ftmPeriod is stored in the state structure.
in	<i>softwareTrigger</i>	If true a software trigger is generate to update PWM parameters.

#### Returns

success

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 447 of file ftm\_pwm\_driver.c.

**16.60.5.6** `status_t FTM_DRV_UpdatePwmPeriod ( uint32_t instance, ftm_pwm_update_option_t typeOfUpdate, uint32_t newValue, bool softwareTrigger )`

This function will update the new period in the frequency or in the counter value into mode register which modify the period of PWM signal on the channel output.

#### Parameters

in	<i>instance</i>	The FTM peripheral instance number.
in	<i>typeOfUpdate</i>	The type of PWM update is a period in Hz or in ticks. <ul style="list-style-type: none"> <li>• For FTM_PWM_UPDATE_IN_DUTY_CYCLE which reuse in FTM_DRV_UpdatePwmChannel function will update in Hz.</li> <li>• For FTM_PWM_UPDATE_IN_TICKS will update in ticks.</li> </ul>
in	<i>newValue</i>	The frequency or the counter value which will select with modified value for PWM signal. If the type of update in the duty cycle, the newValue parameter must be value between 1U and maximum is the frequency of the FTM counter. If the type of update in ticks, the newValue parameter must be value between 1U and 0xFFFFU.



in	<i>softwareTrigger</i>	If true a software trigger is generate to update PWM parameters.
----	------------------------	--

**Returns**

operation status

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 576 of file ftm\_pwm\_driver.c.

## 16.61 FlexTimer Quadrature Decoder Driver (FTM\_QD)

### 16.61.1 Detailed Description

FlexTimer Quadrature Decoder Peripheral Driver.

#### Hardware background

The FTM of the S32K1xx is based on a 16 bits counter and supports: input capture, output compare, PWM and some instances include quadrature decoder.

#### How to use FTM driver in your application

For all operation modes (without Quadrature Decoder mode) the user need to configure [ftm\\_user\\_config\\_t](#). This structure will be used for initialization (FTM\_DRV\_Init). The next functions used are specific for each operation mode.

#### Quadrature decoder mode

For this mode the user needs to configure parameters like: maximum counter value, initial counter value, mode (Count and Direction Encoding mode), and for both input phases polarity and filtering. All this information is included in [ftm\\_quad\\_decode\\_config\\_t](#). In this mode, the counter is clocked by the phase A and phase B. The current state of the decoder can be obtained using FTM\_DRV\_QuadGetState.

#### Hardware limitation:

In count and direction mode if initial value of the PHASE\_A is HIGH the counter will be incremented.

#### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_qd_driver.c
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_common.c
{S32SDK_PATH}\platform\drivers\src\ftm\ftm_hw_access.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
{S32SDK_PATH}\platform\drivers\inc\
{S32SDK_PATH}\platform\drivers\src\ftm\
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

#### [Clock Manager Interrupt Manager \(Interrupt\)](#)

Example:

```
/* The state structure of instance in the quadrature mode */
ftm_state_t stateQuad;
#define FTM_QUADRATURE_INSTANCE 1UL
ftm_quad_decoder_state_t quadra_state;
ftm_quad_decode_config_t quadrature_decoder_configuration =
{
    FTM_QUAD_COUNT_AND_DIR, /* Quadrature decoder mode */
    0U, /* Initial counter value */
    32500U, /* Maximum counter value */
    {
```

```

        false,                /* Filter state */
        0U,                  /* Filter value */
        FTM_QUAD_PHASE_NORMAL /* Phase polarity */
    },
    {
        false,                /* Filter state */
        0U,                  /* Filter value */
        FTM_QUAD_PHASE_NORMAL /* Phase polarity */
    }
};

/* Timer mode configuration for Quadrature */
/* Global configuration of Quadrature */
ftm_user_config_t Quadrature_InitConfig =
{
    {
        false,                /* Software trigger state */
        false,                /* Hardware trigger 1 state */
        false,                /* Hardware trigger 2 state */
        false,                /* Hardware trigger 3 state */
        false,                /* Maximum loading point state */
        false,                /* Min loading point state */
        FTM_SYSTEM_CLOCK,     /* Update mode for INVCTRL register */
        FTM_SYSTEM_CLOCK,     /* Update mode for SWOCTRL register */
        FTM_SYSTEM_CLOCK,     /* Update mode for OUTMASK register */
        FTM_SYSTEM_CLOCK,     /* Update mode for CNTIN register */
        false,                /* Auto clear trigger state for hardware trigger */
        FTM_UPDATE_NOW,       /* Select synchronization method */
    },
    FTM_MODE_QUADRATURE_DECODER, /* Mode of operation for FTM */
    FTM_CLOCK_DIVID_BY_2,       /* FTM clock pre-scaler */
    FTM_CLOCK_SOURCE_SYSTEMCLK, /* FTM clock source */
    FTM_BDM_MODE_11,           /* FTM debug mode */
    false,                     /* Interrupt state */
    false                       /* Initialization trigger */
};

FTM_DRV_Init(FTM_QUADRATURE_INSTANCE, &Quadrature_InitConfig, &stateQuad);
FTM_DRV_QuadDecodeStart(FTM_QUADRATURE_INSTANCE, &quadrature_decoder_configuration);
quadra_state = FTM_DRV_QuadGetState(FTM_QUADRATURE_INSTANCE);

```

## Data Structures

- struct [ftm\\_phase\\_params\\_t](#)  
*FlexTimer quadrature decoder channel parameters. [More...](#)*
- struct [ftm\\_quad\\_decode\\_config\\_t](#)  
*FTM quadrature configure structure. [More...](#)*
- struct [ftm\\_quad\\_decoder\\_state\\_t](#)  
*FTM quadrature state(counter value and flags) [More...](#)*

## Enumerations

- enum [ftm\\_quad\\_decode\\_mode\\_t](#) { FTM\_QUAD\_PHASE\_ENCODE = 0x00U, FTM\_QUAD\_COUNT\_AND\_DIRECTION = 0x01U }  
*FlexTimer quadrature decode modes, phase encode or count and direction mode.*
- enum [ftm\\_quad\\_phase\\_polarity\\_t](#) { FTM\_QUAD\_PHASE\_NORMAL = 0x00U, FTM\_QUAD\_PHASE\_INVERTED = 0x01U }  
*FlexTimer quadrature phase polarities, normal or inverted polarity.*

## Functions

- status\_t [FTM\\_DRV\\_QuadDecodeStart](#) (uint32\_t instance, const [ftm\\_quad\\_decode\\_config\\_t](#) \*config)  
*Configures the quadrature mode and starts measurement.*
- status\_t [FTM\\_DRV\\_QuadDecodeStop](#) (uint32\_t instance)  
*De-activates the quadrature decode mode.*
- [ftm\\_quad\\_decoder\\_state\\_t](#) [FTM\\_DRV\\_QuadGetState](#) (uint32\_t instance)  
*Return the current quadrature decoder state (counter value, overflow flag and overflow direction)*
- void [FTM\\_QD\\_DRV\\_GetDefaultConfig](#) ([ftm\\_quad\\_decode\\_config\\_t](#) \*const config)  
*This function will get the default configuration values in the structure which is used as a common use-case.*

## 16.61.2 Data Structure Documentation

### 16.61.2.1 struct `ftm_phase_params_t`

FlexTimer quadrature decoder channel parameters.

Implements : `ftm_phase_params_t_Class`

Definition at line 64 of file `ftm_qd_driver.h`.

#### Data Fields

- bool [phaseInputFilter](#)
- uint8\_t [phaseFilterVal](#)
- [ftm\\_quad\\_phase\\_polarity\\_t](#) [phasePolarity](#)

#### Field Documentation

##### 16.61.2.1.1 uint8\_t `phaseFilterVal`

Filter value (if input filter is enabled)

Definition at line 68 of file `ftm_qd_driver.h`.

##### 16.61.2.1.2 bool `phaseInputFilter`

false: disable phase filter, true: enable phase filter

Definition at line 66 of file `ftm_qd_driver.h`.

##### 16.61.2.1.3 [ftm\\_quad\\_phase\\_polarity\\_t](#) `phasePolarity`

Phase polarity

Definition at line 69 of file `ftm_qd_driver.h`.

### 16.61.2.2 struct `ftm_quad_decode_config_t`

FTM quadrature configure structure.

Implements : `ftm_quad_decode_config_t_Class`

Definition at line 77 of file `ftm_qd_driver.h`.

#### Data Fields

- [ftm\\_quad\\_decode\\_mode\\_t](#) `mode`
- uint16\_t `initialVal`
- uint16\_t `maxVal`
- [ftm\\_phase\\_params\\_t](#) `phaseAConfig`
- [ftm\\_phase\\_params\\_t](#) `phaseBConfig`

#### Field Documentation

##### 16.61.2.2.1 uint16\_t `initialVal`

Initial counter value

Definition at line 80 of file `ftm_qd_driver.h`.

##### 16.61.2.2.2 uint16\_t `maxVal`

Maximum counter value

Definition at line 81 of file `ftm_qd_driver.h`.

#### 16.61.2.2.3 `ftm_quad_decode_mode_t` mode

FTM\_QUAD\_PHASE\_ENCODE or FTM\_QUAD\_COUNT\_AND\_DIR

Definition at line 79 of file `ftm_qd_driver.h`.

#### 16.61.2.2.4 `ftm_phase_params_t` phaseAConfig

Configuration for the input phase a

Definition at line 82 of file `ftm_qd_driver.h`.

#### 16.61.2.2.5 `ftm_phase_params_t` phaseBConfig

Configuration for the input phase b

Definition at line 83 of file `ftm_qd_driver.h`.

#### 16.61.2.3 `struct ftm_quad_decoder_state_t`

FTM quadrature state(counter value and flags)

Implements : `ftm_quad_decoder_state_t_Class`

Definition at line 91 of file `ftm_qd_driver.h`.

##### Data Fields

- `uint16_t` [counter](#)
- `bool` [overflowFlag](#)
- `bool` [overflowDirection](#)
- `bool` [counterDirection](#)

##### Field Documentation

#### 16.61.2.3.1 `uint16_t` counter

Counter value

Definition at line 93 of file `ftm_qd_driver.h`.

#### 16.61.2.3.2 `bool` counterDirection

False FTM counter is decreasing, True FTM counter is increasing

Definition at line 98 of file `ftm_qd_driver.h`.

#### 16.61.2.3.3 `bool` overflowDirection

False if overflow occurred at minimum value, True if overflow occurred at maximum value

Definition at line 96 of file `ftm_qd_driver.h`.

#### 16.61.2.3.4 `bool` overflowFlag

True if overflow occurred, False if overflow doesn't occurred

Definition at line 94 of file `ftm_qd_driver.h`.

### 16.61.3 Enumeration Type Documentation

#### 16.61.3.1 `enum ftm_quad_decode_mode_t`

FlexTimer quadrature decode modes, phase encode or count and direction mode.

Implements : `ftm_quad_decode_mode_t_Class`

#### Enumerator

**FTM\_QUAD\_PHASE\_ENCODE** Phase encoding mode

**FTM\_QUAD\_COUNT\_AND\_DIR** Counter and direction encoding mode

Definition at line 40 of file `ftm_qd_driver.h`.

#### 16.61.3.2 enum `ftm_quad_phase_polarity_t`

FlexTimer quadrature phase polarities, normal or inverted polarity.

Implements : `ftm_quad_phase_polarity_t_Class`

#### Enumerator

**FTM\_QUAD\_PHASE\_NORMAL** Phase input signal is not inverted before identifying the rising and falling edges of this signal

**FTM\_QUAD\_PHASE\_INVERT** Phase input signal is inverted before identifying the rising and falling edges of this signal

Definition at line 51 of file `ftm_qd_driver.h`.

### 16.61.4 Function Documentation

#### 16.61.4.1 `status_t FTM_DRV_QuadDecodeStart ( uint32_t instance, const ftm_quad_decode_config_t * config )`

Configures the quadrature mode and starts measurement.

##### Parameters

<code>in</code>	<code>instance</code>	Instance number of the FTM module.
<code>in</code>	<code>config</code>	Configuration structure(quadrature decode mode, polarity for both phases, initial and maximum value for the counter, filter configuration).

##### Returns

success

- `STATUS_SUCCESS` : Completed successfully.
- `STATUS_ERROR` : Error occurred.

Definition at line 47 of file `ftm_qd_driver.c`.

#### 16.61.4.2 `status_t FTM_DRV_QuadDecodeStop ( uint32_t instance )`

De-activates the quadrature decode mode.

##### Parameters

<code>in</code>	<code>instance</code>	Instance number of the FTM module.
-----------------	-----------------------	------------------------------------

##### Returns

success

- `STATUS_SUCCESS` : Completed successfully.
- `STATUS_ERROR` : Error occurred.

Definition at line 106 of file `ftm_qd_driver.c`.

#### 16.61.4.3 `ftm_quad_decoder_state_t FTM_DRV_QuadGetState ( uint32_t instance )`

Return the current quadrature decoder state (counter value, overflow flag and overflow direction)

**Parameters**

<i>in</i>	<i>instance</i>	Instance number of the FTM module.
-----------	-----------------	------------------------------------

**Returns**

The current state of quadrature decoder

Definition at line 128 of file ftm\_qd\_driver.c.

**16.61.4.4 void FTM\_QD\_DRV\_GetDefaultConfig ( ftm\_quad\_decode\_config\_t \*const config )**

This function will get the default configuration values in the structure which is used as a common use-case.

**Parameters**

<i>out</i>	<i>config</i>	Pointer to the structure in which the configuration will be saved.
------------	---------------	--

**Returns**

None

Definition at line 150 of file ftm\_qd\_driver.c.

## 16.62 Flexible I/O (FlexIO)

### 16.62.1 Detailed Description

The FlexIO is a highly configurable module providing a wide range of functionality including:

- Emulation of a variety of serial communication protocols, such as SPI, I2C, I2S or UART, while requiring low CPU overhead and being more efficient than having multiple dedicated peripherals for each protocol.
- Flexible 16-bit timers with support for a variety of trigger, reset, enable and disable conditions
- PWM/Waveform generation

Several drivers are provided for this device, implementing a variety of communication protocols. There is also a common layer on which all the drivers are based, allowing more driver instances, either of the same type or different types, to function in parallel on the same FlexIO device. Each driver instance needs a certain number of FlexIO resources (shifters and timers) and as long as there are enough free resources new driver instances can be initialized. The table below shows the driver types and the number of resources needed by each one:

Drivers	Timers	Shifters	Pins
SPI	2	2	4
I2C	2	2	2
I2S	2	2	4
UART	1	1	1

The number of timers and shifters available on a specific device can be found in the reference manual.

### Modules

- [FlexIO Common Driver](#)  
*Common services for FlexIO drivers.*
- [FlexIO I2C Driver](#)  
*I2C communication over FlexIO module (FLEXIO\_I2C)*
- [FlexIO I2S Driver](#)  
*I2S communication over FlexIO module (FLEXIO\_I2S)*
- [FlexIO SPI Driver](#)  
*SPI communication over FlexIO module (FLEXIO\_SPI)*
- [FlexIO UART Driver](#)  
*UART communication over FlexIO module (FLEXIO\_UART)*



## 16.63 FreeRTOS

FreeRTOS is a Real Time Operating System (RTOS) design to run on microcontrollers which have size constraints and dedicated end applications.

FreeRTOS provides:

- core real time scheduling functionality
- inter-task communication
- timing and synchronisation primitives

Additional functionality can be included with add-on components.

More information about FreeRTOS can be found on the FreeRTOS website: [www.freertos.org](http://www.freertos.org)

### Compiler Settings

Please refer to the "Compiler options" section in Release Notes document when creating a new project. The provided compiler options must match with the project to avoid compilation issue or undefined behavior.

## 16.64 HVIO high-side configuration options (0x71)

### 16.64.1 Detailed Description

This section lists all the possible options of 0x71 register related to:

- HVIO N high-side driver configurations

See also

[UJA113X\\_SetMtpnv\(\)](#)

#### HVIO8 high-side driver configuration options (0x71:IO8HOC[7])

- #define [HHSDC\\_IO8HOC\\_ENABLED](#) 0x00U  
*HVIO8 high-side driver is controlled by the setting in the HVIO8 control register (See [uja113x\\_hvio\\_cfg\\_t](#))*
- #define [HHSDC\\_IO8HOC\\_DISABLED](#) 0x80U  
*HVIO8 high-side driver is off, regardless of the setting in the HVIO8 control register.*

#### HVIO7 high-side driver configuration options (0x71:IO7HOC[6])

- #define [HHSDC\\_IO7HOC\\_ENABLED](#) 0x00U  
*HVIO7 high-side driver is controlled by the setting in the HVIO7 control register (See [uja113x\\_hvio\\_cfg\\_t](#))*
- #define [HHSDC\\_IO7HOC\\_DISABLED](#) 0x40U  
*HVIO7 high-side driver is off, regardless of the setting in the HVIO7 control register.*

#### HVIO6 high-side driver configuration options (0x71:IO6HOC[5])

- #define [HHSDC\\_IO6HOC\\_ENABLED](#) 0x00U  
*HVIO6 high-side driver is controlled by the setting in the HVIO6 control register (See [uja113x\\_hvio\\_cfg\\_t](#))*
- #define [HHSDC\\_IO6HOC\\_DISABLED](#) 0x20U  
*HVIO6 high-side driver is off, regardless of the setting in the HVIO6 control register.*

#### HVIO5 high-side driver configuration options (0x71:IO5HOC[4])

- #define [HHSDC\\_IO5HOC\\_ENABLED](#) 0x00U  
*HVIO5 high-side driver is controlled by the setting in the HVIO5 control register (See [uja113x\\_hvio\\_cfg\\_t](#))*
- #define [HHSDC\\_IO5HOC\\_DISABLED](#) 0x10U  
*HVIO5 high-side driver is off, regardless of the setting in the HVIO5 control register.*

#### HVIO4 high-side driver configuration options (0x71:IO4HOC[3])

- #define [HHSDC\\_IO4HOC\\_ENABLED](#) 0x00U  
*HVIO4 high-side driver is controlled by the setting in the HVIO4 control register (See [uja113x\\_hvio\\_cfg\\_t](#))*
- #define [HHSDC\\_IO4HOC\\_DISABLED](#) 0x08U  
*HVIO4 high-side driver is off, regardless of the setting in the HVIO4 control register.*

#### HVIO3 high-side driver configuration options (0x71:IO3HOC[2])

- #define [HHSDC\\_IO3HOC\\_ENABLED](#) 0x00U  
*HVIO3 high-side driver is controlled by the setting in the HVIO3 control register (See [uja113x\\_hvio\\_cfg\\_t](#))*
- #define [HHSDC\\_IO3HOC\\_DISABLED](#) 0x04U  
*HVIO3 high-side driver is off, regardless of the setting in the HVIO3 control register.*

## HVIO2 high-side driver configuration options (0x71:IO2HOC[1])

- #define `HHSDC_IO2HOC_ENABLED` 0x00U  
HVIO2 high-side driver is controlled by the setting in the HVIO2 control register (See [uja113x\\_hvio\\_cfg\\_t](#))
- #define `HHSDC_IO2HOC_DISABLED` 0x02U  
HVIO2 high-side driver is off, regardless of the setting in the HVIO2 control register.

## HVIO1 high-side driver configuration options (0x71:IO1HOC[0])

- #define `HHSDC_IO1HOC_ENABLED` 0x00U  
HVIO1 high-side driver is controlled by the setting in the HVIO1 control register (See [uja113x\\_hvio\\_cfg\\_t](#))
- #define `HHSDC_IO1HOC_DISABLED` 0x01U  
HVIO1 high-side driver is off, regardless of the setting in the HVIO1 control register.

## 16.64.2 Macro Definition Documentation

16.64.2.1 #define `HHSDC_IO1HOC_DISABLED` 0x01U

HVIO1 high-side driver is off, regardless of the setting in the HVIO1 control register.

Definition at line 1187 of file `sbc_uja113x.h`.

16.64.2.2 #define `HHSDC_IO1HOC_ENABLED` 0x00U

HVIO1 high-side driver is controlled by the setting in the HVIO1 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1186 of file `sbc_uja113x.h`.

16.64.2.3 #define `HHSDC_IO2HOC_DISABLED` 0x02U

HVIO2 high-side driver is off, regardless of the setting in the HVIO2 control register.

Definition at line 1180 of file `sbc_uja113x.h`.

16.64.2.4 #define `HHSDC_IO2HOC_ENABLED` 0x00U

HVIO2 high-side driver is controlled by the setting in the HVIO2 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1179 of file `sbc_uja113x.h`.

16.64.2.5 #define `HHSDC_IO3HOC_DISABLED` 0x04U

HVIO3 high-side driver is off, regardless of the setting in the HVIO3 control register.

Definition at line 1173 of file `sbc_uja113x.h`.

16.64.2.6 #define `HHSDC_IO3HOC_ENABLED` 0x00U

HVIO3 high-side driver is controlled by the setting in the HVIO3 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1172 of file `sbc_uja113x.h`.

16.64.2.7 #define `HHSDC_IO4HOC_DISABLED` 0x08U

HVIO4 high-side driver is off, regardless of the setting in the HVIO4 control register.

Definition at line 1166 of file `sbc_uja113x.h`.

16.64.2.8 #define `HHSDC_IO4HOC_ENABLED` 0x00U

HVIO4 high-side driver is controlled by the setting in the HVIO4 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1165 of file `sbc_uja113x.h`.

**16.64.2.9 #define HHSDC\_IO5HOC\_DISABLED 0x10U**

HVIO5 high-side driver is off, regardless of the setting in the HVIO5 control register.

Definition at line 1159 of file sbc\_uja113x.h.

**16.64.2.10 #define HHSDC\_IO5HOC\_ENABLED 0x00U**

HVIO5 high-side driver is controlled by the setting in the HVIO5 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1158 of file sbc\_uja113x.h.

**16.64.2.11 #define HHSDC\_IO6HOC\_DISABLED 0x20U**

HVIO6 high-side driver is off, regardless of the setting in the HVIO6 control register.

Definition at line 1152 of file sbc\_uja113x.h.

**16.64.2.12 #define HHSDC\_IO6HOC\_ENABLED 0x00U**

HVIO6 high-side driver is controlled by the setting in the HVIO6 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1151 of file sbc\_uja113x.h.

**16.64.2.13 #define HHSDC\_IO7HOC\_DISABLED 0x40U**

HVIO7 high-side driver is off, regardless of the setting in the HVIO7 control register.

Definition at line 1145 of file sbc\_uja113x.h.

**16.64.2.14 #define HHSDC\_IO7HOC\_ENABLED 0x00U**

HVIO7 high-side driver is controlled by the setting in the HVIO7 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1144 of file sbc\_uja113x.h.

**16.64.2.15 #define HHSDC\_IO8HOC\_DISABLED 0x80U**

HVIO8 high-side driver is off, regardless of the setting in the HVIO8 control register.

Definition at line 1138 of file sbc\_uja113x.h.

**16.64.2.16 #define HHSDC\_IO8HOC\_ENABLED 0x00U**

HVIO8 high-side driver is controlled by the setting in the HVIO8 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1137 of file sbc\_uja113x.h.

## 16.65 HVIO low-side configuration options (0x72)

### 16.65.1 Detailed Description

This section lists all the possible options of 0x72 register related to:

- HVIO N low-side driver configurations

See also

[UJA113X\\_SetMtpnv\(\)](#)

#### HVIO8 low-side driver configuration options (0x72:IO8LOC[7])

- #define [HLSDC\\_IO8LOC\\_ENABLED](#) 0x00U  
*HVIO8 low-side driver is controlled by the setting in the HVIO8 control register (See [uja113x\\_hvio\\_cfg\\_t](#))*
- #define [HLSDC\\_IO8LOC\\_DISABLED](#) 0x80U  
*HVIO8 low-side driver is off, regardless of the setting in the HVIO8 control register.*

#### HVIO7 low-side driver configuration options (0x72:IO7LOC[6])

- #define [HLSDC\\_IO7LOC\\_ENABLED](#) 0x00U  
*HVIO7 low-side driver is controlled by the setting in the HVIO8 control register (See [uja113x\\_hvio\\_cfg\\_t](#))*
- #define [HLSDC\\_IO7LOC\\_DISABLED](#) 0x40U  
*HVIO4 low-side driver is off, regardless of the setting in the HVIO8 control register.*

#### HVIO6 low-side driver configuration options (0x72:IO6LOC[5])

- #define [HLSDC\\_IO6LOC\\_ENABLED](#) 0x00U  
*HVIO6 low-side driver is controlled by the setting in the HVIO6 control register (See [uja113x\\_hvio\\_cfg\\_t](#))*
- #define [HLSDC\\_IO6LOC\\_DISABLED](#) 0x20U  
*HVIO6 low-side driver is off, regardless of the setting in the HVIO6 control register.*

#### HVIO5 low-side driver configuration options (0x72:IO5LOC[4])

- #define [HLSDC\\_IO5LOC\\_ENABLED](#) 0x00U  
*HVIO5 low-side driver is controlled by the setting in the HVIO5 control register (See [uja113x\\_hvio\\_cfg\\_t](#))*
- #define [HLSDC\\_IO5LOC\\_DISABLED](#) 0x10U  
*HVIO5 low-side driver is off, regardless of the setting in the HVIO5 control register.*

#### HVIO4 low-side driver configuration options (0x72:IO4LOC[3])

- #define [HLSDC\\_IO4LOC\\_ENABLED](#) 0x00U  
*HVIO4 low-side driver is controlled by the setting in the HVIO4 control register (See [uja113x\\_hvio\\_cfg\\_t](#))*
- #define [HLSDC\\_IO4LOC\\_DISABLED](#) 0x08U  
*HVIO4 low-side driver is off, regardless of the setting in the HVIO4 control register.*

#### HVIO3 low-side driver configuration options (0x72:IO3LOC[2])

- #define [HLSDC\\_IO3LOC\\_ENABLED](#) 0x00U  
*HVIO3 low-side driver is controlled by the setting in the HVIO3 control register (See [uja113x\\_hvio\\_cfg\\_t](#))*
- #define [HLSDC\\_IO3LOC\\_DISABLED](#) 0x04U  
*HVIO3 low-side driver is off, regardless of the setting in the HVIO3 control register.*

### HVIO2 low-side driver configuration options (0x72:IO2LOC[1])

- `#define HLSDC_IO2LOC_ENABLED 0x00U`  
*HVIO2 low-side driver is controlled by the setting in the HVIO2 control register (See [uja113x\\_hvio\\_cfg\\_t](#))*
- `#define HLSDC_IO2LOC_DISABLED 0x02U`  
*HVIO2 low-side driver is off, regardless of the setting in the HVIO2 control register.*

### HVIO1 low-side driver configuration options (0x72:IO1LOC[0])

- `#define HLSDC_IO1LOC_ENABLED 0x00U`  
*HVIO1 low-side driver is controlled by the setting in the HVIO1 control register (See [uja113x\\_hvio\\_cfg\\_t](#))*
- `#define HLSDC_IO1LOC_DISABLED 0x01U`  
*HVIO1 low-side driver is off, regardless of the setting in the HVIO1 control register.*

## 16.65.2 Macro Definition Documentation

### 16.65.2.1 `#define HLSDC_IO1LOC_DISABLED 0x01U`

HVIO1 low-side driver is off, regardless of the setting in the HVIO1 control register.

Definition at line 1256 of file `sbc_uja113x.h`.

### 16.65.2.2 `#define HLSDC_IO1LOC_ENABLED 0x00U`

HVIO1 low-side driver is controlled by the setting in the HVIO1 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1255 of file `sbc_uja113x.h`.

### 16.65.2.3 `#define HLSDC_IO2LOC_DISABLED 0x02U`

HVIO2 low-side driver is off, regardless of the setting in the HVIO2 control register.

Definition at line 1249 of file `sbc_uja113x.h`.

### 16.65.2.4 `#define HLSDC_IO2LOC_ENABLED 0x00U`

HVIO2 low-side driver is controlled by the setting in the HVIO2 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1248 of file `sbc_uja113x.h`.

### 16.65.2.5 `#define HLSDC_IO3LOC_DISABLED 0x04U`

HVIO3 low-side driver is off, regardless of the setting in the HVIO3 control register.

Definition at line 1242 of file `sbc_uja113x.h`.

### 16.65.2.6 `#define HLSDC_IO3LOC_ENABLED 0x00U`

HVIO3 low-side driver is controlled by the setting in the HVIO3 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1241 of file `sbc_uja113x.h`.

### 16.65.2.7 `#define HLSDC_IO4LOC_DISABLED 0x08U`

HVIO4 low-side driver is off, regardless of the setting in the HVIO4 control register.

Definition at line 1235 of file `sbc_uja113x.h`.

### 16.65.2.8 `#define HLSDC_IO4LOC_ENABLED 0x00U`

HVIO4 low-side driver is controlled by the setting in the HVIO4 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1234 of file `sbc_uja113x.h`.

**16.65.2.9 #define HLSDC\_IO5LOC\_DISABLED 0x10U**

HVIO5 low-side driver is off, regardless of the setting in the HVIO5 control register.

Definition at line 1228 of file sbc\_uja113x.h.

**16.65.2.10 #define HLSDC\_IO5LOC\_ENABLED 0x00U**

HVIO5 low-side driver is controlled by the setting in the HVIO5 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1227 of file sbc\_uja113x.h.

**16.65.2.11 #define HLSDC\_IO6LOC\_DISABLED 0x20U**

HVIO6 low-side driver is off, regardless of the setting in the HVIO6 control register.

Definition at line 1221 of file sbc\_uja113x.h.

**16.65.2.12 #define HLSDC\_IO6LOC\_ENABLED 0x00U**

HVIO6 low-side driver is controlled by the setting in the HVIO6 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1220 of file sbc\_uja113x.h.

**16.65.2.13 #define HLSDC\_IO7LOC\_DISABLED 0x40U**

HVIO4 low-side driver is off, regardless of the setting in the HVIO8 control register.

Definition at line 1214 of file sbc\_uja113x.h.

**16.65.2.14 #define HLSDC\_IO7LOC\_ENABLED 0x00U**

HVIO7 low-side driver is controlled by the setting in the HVIO8 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1213 of file sbc\_uja113x.h.

**16.65.2.15 #define HLSDC\_IO8LOC\_DISABLED 0x80U**

HVIO8 low-side driver is off, regardless of the setting in the HVIO8 control register.

Definition at line 1207 of file sbc\_uja113x.h.

**16.65.2.16 #define HLSDC\_IO8LOC\_ENABLED 0x00U**

HVIO8 low-side driver is controlled by the setting in the HVIO8 control register (See [uja113x\\_hvio\\_cfg\\_t](#))

Definition at line 1206 of file sbc\_uja113x.h.

## 16.66 I2S - Peripheral Abstraction Layer (I2S PAL)

### 16.66.1 Detailed Description

The S32 SDK provides a Peripheral Abstraction Layer for I2S modules of S32 SDK devices.

The I2S PAL is designed to be portable across all platforms and IPs which support I2S communication.

#### Integration guideline

##### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\pal\src\i2s\i2s_pal.c
```

##### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\pal\inc  
i2s_pal_cfg.h path (this file is provided by user or generated by pex)
```

##### Compile symbols

No special symbols are required for this component

##### Dependencies

sai flexio\_i2s [Enhanced Direct Memory Access \(eDMA\)](#) [OS Interface \(OSIF\)](#) [Clock manager](#) [Interrupt Manager \(Interrupt\)](#)

#### How to integrate I2S PAL in your application

Unlike the other drivers, I2S PAL modules need to include a configuration file named i2s\_pal\_cfg.h, which allows the user to specify which IPs are used and how many resources are allocated for each of them (state structures). The following code example shows how to configure one instance for each available I2S IPs.

```
#ifndef i2s_pal_cfg_H  
#define i2s_pal_cfg_H  
  
/* Define which IP instance will be used in current project */  
#define I2S_OVER_FLEXIO  
#define I2S_OVER_SAI  
  
/* Define the resources necessary for current project */  
#define NO_OF_FLEXIO_MASTER_INSTS_FOR_I2S 1U  
#define NO_OF_FLEXIO_SLAVE_INSTS_FOR_I2S 1U  
#define NO_OF_SAI_INSTS_FOR_I2S 1U  
  
#endif /* i2s_pal_cfg_H */
```

The following table contains the matching between platforms and available IPs

IP/MCU	S32K116	S32K118	S32K142	S32K144	S32K146	S32K148	MP↔ C5748G	MP↔ C5746C
SAI	NO	NO	NO	NO	NO	YES	YES	YES
FLEXIO↔ O_I2S	YES	YES	YES	YES	YES	YES	NO	NO

In order to use the I2S driver it must be first initialized using function I2S\_Init. Once initialized, it cannot be initialized again for the same instance until it is de-initialized, using I2S\_Deinit. Different instances can work independently of each other.

#### Important Notes

- I2S PAL only works in half duplex mode, meaning it cannot send and receive simultaneously over one instance.



- The driver enables the interrupts for the corresponding module, but any interrupt priority setting must be done by the application.
- When using SAI module for I2S, a success status of sending operation means that all data has been pushed to hardware fifo (not output pin), up to 8 tx words will be discarded if users call deinit or switch to receiving operation right after that.

### Example code

```

/* Buffers */
uint8_t tx[6] = {1,2,3,4,5,6};

/* Callback to continue sending data */
void i2s_Callback0(i2s_event_t event, void *userData)
{
    /* Get from userData the I2S instance of the I2S module */
    i2s_instance_t* instance;
    instance = (i2s_instance_t*)(userData);

    /* Check the event type:
     * - set TX buffers on I2S_EVENT_TX_EMPTY
     */
    if (event == I2S_EVENT_TX_EMPTY)
    {
        I2S_SetTxBuffer(instance, tx, 6UL);
    }
}

/* Define I2S instance */
i2s_instance_t flexioInst = {I2S_INST_TYPE_FLEXIO, 0UL};
/* Configure I2S */
i2s_user_config_t i2sUserConfig0 =
{
    .baudRate      = 1000000U,
    .mode          = I2S_MASTER,
    .wordWidth     = 8U,
    .transferType  = I2S_USING_INTERRUPT,
    .rxDMACHannel  = 0U,
    .txDMACHannel  = 0U,
    .callback      = i2s_Callback0,
    .callbackParam = &flexioInst,
    .extension     = NULL
};

/* Configure FLEXIO pins routing */
extension_flexio_for_i2s_t extension;
extension.txPin = 0U;
extension.rxPin = 1U;
extension.sckPin = 2U;
extension.wsPin = 3U;
i2sUserConfig0.extension = &extension;

/* Initializes i2s master for flexio 0 and send 6 words */
I2S_Init(&flexioInst, &i2sUserConfig0);
I2S_SendData(&flexioInst, tx, 6UL);
/* Wait for sending complete */
while (I2S_GetStatus(&flexioInst, NULL) == STATUS_BUSY);
I2S_Deinit(&flexioInst);

```

### Data Structures

- struct [i2s\\_user\\_config\\_t](#)  
*I2S user configuration structure. [More...](#)*
- struct [extension\\_flexio\\_for\\_i2s\\_t](#)  
*Defines the extension structure for the I2S over FLEXIO. [More...](#)*

### Enumerations

- enum [i2s\\_transfer\\_type\\_t](#) { [I2S\\_USING\\_INTERRUPT](#) = 0U, [I2S\\_USING\\_DMA](#) = 1U }  
*Defines the transfer type.*
- enum [i2s\\_mode\\_t](#) { [I2S\\_MASTER](#) = 0U, [I2S\\_SLAVE](#) = 1U }  
*Master or slave.*

## Functions

- [status\\_t I2S\\_Init](#) (const [i2s\\_instance\\_t](#) \*instance, const [i2s\\_user\\_config\\_t](#) \*config)  
*Initializes the I2S module.*
- [status\\_t I2S\\_Deinit](#) (const [i2s\\_instance\\_t](#) \*instance)  
*De-initializes the I2S module.*
- [status\\_t I2S\\_GetBaudRate](#) (const [i2s\\_instance\\_t](#) \*instance, [uint32\\_t](#) \*configuredBaudRate)  
*Returns the i2s baud rate.*
- [status\\_t I2S\\_SetTxBuffer](#) (const [i2s\\_instance\\_t](#) \*instance, const [uint8\\_t](#) \*txBuff, [uint32\\_t](#) txSize)  
*Keep sending.*
- [status\\_t I2S\\_SetRxBuffer](#) (const [i2s\\_instance\\_t](#) \*instance, [uint8\\_t](#) \*rxBuff, [uint32\\_t](#) rxSize)  
*Keep receiving.*
- [status\\_t I2S\\_SendDataBlocking](#) (const [i2s\\_instance\\_t](#) \*instance, const [uint8\\_t](#) \*txBuff, [uint32\\_t](#) txSize, [uint32\\_t](#) timeout)  
*Perform a blocking I2S transmission.*
- [status\\_t I2S\\_SendData](#) (const [i2s\\_instance\\_t](#) \*instance, const [uint8\\_t](#) \*txBuff, [uint32\\_t](#) txSize)  
*Perform a non-blocking I2S transmission.*
- [status\\_t I2S\\_GetStatus](#) (const [i2s\\_instance\\_t](#) \*instance, [uint32\\_t](#) \*countRemaining)  
*Get the status of the current I2S transfer.*
- [status\\_t I2S\\_ReceiveDataBlocking](#) (const [i2s\\_instance\\_t](#) \*instance, [uint8\\_t](#) \*rxBuff, [uint32\\_t](#) rxSize, [uint32\\_t](#) timeout)  
*Perform a blocking I2S reception.*
- [status\\_t I2S\\_ReceiveData](#) (const [i2s\\_instance\\_t](#) \*instance, [uint8\\_t](#) \*rxBuff, [uint32\\_t](#) rxSize)  
*Perform a non-blocking I2S reception.*
- [status\\_t I2S\\_Abort](#) (const [i2s\\_instance\\_t](#) \*instance)  
*Terminates a non-blocking transfer early.*
- [void I2S\\_GetDefaultConfig](#) ([i2s\\_user\\_config\\_t](#) \*const config)  
*Return default configuration.*

## 16.66.2 Data Structure Documentation

### 16.66.2.1 [struct i2s\\_user\\_config\\_t](#)

I2S user configuration structure.

Implements : [i2s\\_user\\_config\\_t\\_Class](#)

Definition at line 59 of file [i2s\\_pal.h](#).

## Data Fields

- [i2s\\_transfer\\_type\\_t](#) transferType
- [i2s\\_mode\\_t](#) mode
- [uint32\\_t](#) baudRate
- [uint8\\_t](#) wordWidth
- [i2s\\_callback\\_t](#) callback
- [void \\*](#) callbackParam
- [uint8\\_t](#) rxDMACChannel
- [uint8\\_t](#) txDMACChannel
- [void \\*](#) extension

## Field Documentation

### 16.66.2.1.1 [uint32\\_t](#) baudRate

Baud rate in hertz

Definition at line 63 of file [i2s\\_pal.h](#).

**16.66.2.1.2 i2s\_callback\_t callback**

User callback function. Can be null if not needed.

Definition at line 67 of file i2s\_pal.h.

**16.66.2.1.3 void\* callbackParam**

Parameter for the callback function

Definition at line 68 of file i2s\_pal.h.

**16.66.2.1.4 void\* extension**

This field will be used to add extra settings to the basic configuration like FlexIO data pins

Definition at line 71 of file i2s\_pal.h.

**16.66.2.1.5 i2s\_mode\_t mode**

Master or slave

Definition at line 62 of file i2s\_pal.h.

**16.66.2.1.6 uint8\_t rxDMAChannel**

Rx DMA channel number. Only used in DMA mode

Definition at line 69 of file i2s\_pal.h.

**16.66.2.1.7 i2s\_transfer\_type\_t transferType**

Driver type: interrupts/DMA

Definition at line 61 of file i2s\_pal.h.

**16.66.2.1.8 uint8\_t txDMAChannel**

Tx DMA channel number. Only used in DMA mode

Definition at line 70 of file i2s\_pal.h.

**16.66.2.1.9 uint8\_t wordWidth**

Number of bits in a word - multiple of 8. The word size in transfer functions depends on this parameter Word size for each buffer read/write is 1 byte, 2 bytes or 4 byte, whichever larger and close to wordWidth the most

Definition at line 64 of file i2s\_pal.h.

**16.66.2.2 struct extension\_flexio\_for\_i2s\_t**

Defines the extension structure for the I2S over FLEXIO.

Definition at line 78 of file i2s\_pal.h.

**Data Fields**

- [uint8\\_t txPin](#)
- [uint8\\_t rxPin](#)
- [uint8\\_t sckPin](#)
- [uint8\\_t wsPin](#)

**Field Documentation**

#### 16.66.2.2.1 uint8\_t rxPin

Flexio pin to use for receive

Definition at line 81 of file i2s\_pal.h.

#### 16.66.2.2.2 uint8\_t sckPin

Flexio pin to use for serial clock

Definition at line 82 of file i2s\_pal.h.

#### 16.66.2.2.3 uint8\_t txPin

Flexio pin to use for transmit

Definition at line 80 of file i2s\_pal.h.

#### 16.66.2.2.4 uint8\_t wsPin

Flexio pin to use for word select

Definition at line 83 of file i2s\_pal.h.

### 16.66.3 Enumeration Type Documentation

#### 16.66.3.1 enum i2s\_mode\_t

Master or slave.

Implements : i2s\_mode\_t\_Class

Enumerator

**I2S\_MASTER** Generate bit clock and word select signal

**I2S\_SLAVE** Receive bit clock and word select signal

Definition at line 48 of file i2s\_pal.h.

#### 16.66.3.2 enum i2s\_transfer\_type\_t

Defines the transfer type.

Implements : i2s\_transfer\_type\_t\_Class

Enumerator

**I2S\_USING\_INTERRUPT** Driver uses interrupts for data transfers

**I2S\_USING\_DMA** Driver uses DMA for data transfers

Definition at line 37 of file i2s\_pal.h.

### 16.66.4 Function Documentation

#### 16.66.4.1 status\_t I2S\_Abort ( const i2s\_instance\_t \* instance )

Terminates a non-blocking transfer early.

## Parameters

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

Definition at line 694 of file i2s\_pal.c.

#### 16.66.4.2 status\_t I2S\_Deinit ( const i2s\_instance\_t \* *instance* )

De-initializes the I2S module.

This function de-initializes the I2S module.

## Parameters

<i>in</i>	<i>instance</i>	Instance number
-----------	-----------------	-----------------

Definition at line 453 of file i2s\_pal.c.

#### 16.66.4.3 status\_t I2S\_GetBaudRate ( const i2s\_instance\_t \* *instance*, uint32\_t \* *configuredBaudRate* )

Returns the i2s baud rate.

This function returns the i2s configured baud rate, only call this when instance is configured as master.

## Parameters

	<i>instance</i>	Instance number.
<i>out</i>	<i>configured↔ BaudRate</i>	configured baud rate.

## Returns

STATUS\_SUCCESS

Definition at line 494 of file i2s\_pal.c.

#### 16.66.4.4 void I2S\_GetDefaultConfig ( i2s\_user\_config\_t \*const *config* )

Return default configuration.

Return default config for 8 kHz sample rate, 16 bit sample width and 2 channels.

## Parameters

<i>out</i>	<i>Pointer</i>	to configuration structure
------------	----------------	----------------------------

Definition at line 907 of file i2s\_pal.c.

#### 16.66.4.5 status\_t I2S\_GetStatus ( const i2s\_instance\_t \* *instance*, uint32\_t \* *countRemaining* )

Get the status of the current I2S transfer.

## Parameters

<i>instance</i>	Instance number
<i>countRemaining</i>	Pointer to value that is populated with the number of words that have been sent in the active transfer

## Returns

The transmit status.

## Return values

<i>STATUS_SUCCESS</i>	The transmit has completed successfully.
<i>STATUS_BUSY</i>	The transmit is still in progress. will be filled with the number of words that have been transferred so far.
<i>STATUS_I2S_ABORTED</i>	The transmit was aborted.
<i>STATUS_TIMEOUT</i>	A timeout was reached.
<i>STATUS_ERROR</i>	An error occurred.

Definition at line 749 of file i2s\_pal.c.

**16.66.4.6** `status_t I2S_Init ( const i2s_instance_t * instance, const i2s_user_config_t * config )`

Initializes the I2S module.

This function initializes and enables the requested I2S module. Note that when use I2S over SAI, tx and rx line are separated with SAI0, with other SAI instance tx and rx share one line.

## Parameters

in	<i>instance</i>	Instance number
in	<i>config</i>	The configuration structure

Definition at line 294 of file i2s\_pal.c.

**16.66.4.7** `status_t I2S_ReceiveData ( const i2s_instance_t * instance, uint8_t * rxBuff, uint32_t rxSize )`

Perform a non-blocking I2S reception.

This function receives a block of data and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode).

## Parameters

in	<i>instance</i>	Instance number
in	<i>rxBuff</i>	pointer to the data to be transferred
in	<i>rxSize</i>	length in words of the data to be transferred

Definition at line 858 of file i2s\_pal.c.

**16.66.4.8** `status_t I2S_ReceiveDataBlocking ( const i2s_instance_t * instance, uint8_t * rxBuff, uint32_t rxSize, uint32_t timeout )`

Perform a blocking I2S reception.

This function receives a block of data and only returns when the transmission is complete.

## Parameters

in	<i>instance</i>	Instance number
	<i>rxBuff</i>	pointer to the receive buffer
	<i>rxSize</i>	length in words of the data to be received
	<i>timeout</i>	timeout for the transfer in milliseconds

## Returns

Error, success or timed out status

Definition at line 808 of file i2s\_pal.c.

**16.66.4.9** `status_t I2S_SendData ( const i2s_instance_t * instance, const uint8_t * txBuff, uint32_t txSize )`

Perform a non-blocking I2S transmission.

This function sends a block of data and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode).

**Parameters**

in	<i>instance</i>	Instance number
in	<i>txBuff</i>	pointer to the data to be transferred
in	<i>txSize</i>	length in words of the data to be transferred

Definition at line 649 of file i2s\_pal.c.

**16.66.4.10** `status_t I2S_SendDataBlocking ( const i2s_instance_t * instance, const uint8_t * txBuff, uint32_t txSize, uint32_t timeout )`

Perform a blocking I2S transmission.

This function sends a block of data and only returns when the transmission is complete.

**Parameters**

in	<i>instance</i>	Instance number
in	<i>txBuff</i>	pointer to the data to be transferred
in	<i>txSize</i>	length in words of the data to be transferred
in	<i>timeout</i>	timeout value in milliseconds

**Returns**

Error, success or timed out status

Definition at line 526 of file i2s\_pal.c.

**16.66.4.11** `status_t I2S_SetRxBuffer ( const i2s_instance_t * instance, uint8_t * rxBuff, uint32_t rxSize )`

Keep receiving.

This function must be called in callback function when RX\_FULI event is reported to ensure rx operation working continuously.

**Parameters**

in	<i>instance</i>	Instance number
in	<i>rxBuff</i>	pointer to the data to be transferred
in	<i>rxSize</i>	length in words of the data to be transferred

Definition at line 572 of file i2s\_pal.c.

**16.66.4.12** `status_t I2S_SetTxBuffer ( const i2s_instance_t * instance, const uint8_t * txBuff, uint32_t txSize )`

Keep sending.

This function must be called in callback function when TX\_EMPTY event is reported to ensure tx operation working continuously.

**Parameters**

in	<i>instance</i>	Instance number
in	<i>txBuff</i>	pointer to the data to be transferred
in	<i>txSize</i>	length in words of the data to be transferred

Definition at line 611 of file i2s\_pal.c.

## 16.67 Initialization

### 16.67.1 Detailed Description

Initialize transport layer (queues, status, ...).

#### Functions

- void [ld\\_init](#) (l\_ifc\_handle *iii*)  
*Initialize or reinitialize the raw and cooked layers.*

### 16.67.2 Function Documentation

#### 16.67.2.1 void ld\_init ( l\_ifc\_handle *iii* )

Initialize or reinitialize the raw and cooked layers.

##### Parameters

<i>in</i>	<i>iii</i>	Interface name
-----------	------------	----------------

##### Returns

void

Initialize or reinitialize the raw and cooked layers on the interface *iii*. All the transport layer buffers will be initialized.

Definition at line 49 of file `lin_commontl_api.c`.



## 16.68 Input Capture - Peripheral Abstraction Layer (IC PAL)

### 16.68.1 Detailed Description

The S32 SDK provides a Peripheral Abstraction Layer for the input capture mode of S32 SDK devices.

The IC PAL driver allows to detect the input signal and measure pulse width or period of the channel input signal. It was designed to be portable across all platforms and IPs which support FTM , eMIOS, FLEXPWM and ETIMER.

#### How to integrate IC PAL in your application

Unlike the other drivers, IC PAL modules need to include a configuration file named `ic_pal_cfg.h`, which allows the user to specify which IPs are used. The following code example shows how to configure one instance for each available IC IPs.

```
#ifndef IC_PAL_CFG_H
#define IC_PAL_CFG_H

/* Define which IP instance will be used in current project */
#define IC_PAL_OVER_FTM

#endif /* IC_PAL_CFG_H */
```

#### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\pal\src\ic\ic_pal.c
${S32SDK_PATH}\platform\pal\src\ic\ic_irq.c
${S32SDK_PATH}\platform\pal\src\ic\ic_irq.h
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\pal\inc\
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

ftm\_ic emios\_ic etimer flexpwm

The following table contains the matching between platforms and available IPs

IP/ M CU	S32 K116	S32 K118	S32 K142	S32 K144	S32 K146	S32 K148	M P C5748 G	M P C5746 C	M P C5744 P	S32 R274	S32 R372	S32 K142 W	S32 K144 W
FT M _IC	Y ES	Y ES	Y ES	Y ES	Y ES	Y ES	NO	NO	NO	NO	NO	Y ES	Y ES
e MI O S _IC	NO	NO	NO	NO	NO	NO	Y ES	Y ES	NO	NO	NO	NO	NO

E↔ TI↔ M↔ ER	NO	NO	NO	NO	NO	NO	NO	NO	Y↔ ES	Y↔ ES	Y↔ ES	NO	NO
FL↔ E↔ X↔ P↔ WM	NO	NO	NO	NO	NO	NO	NO	NO	Y↔ ES	Y↔ ES	Y↔ ES	NO	NO

## Features

- Start timer channel counting with period in ticks function
- Start/stop the channel in the input capture mode
- Get the measured value in ticks for the detection or measurement

## Functionality

### Initialization

In order to use the IC PAL driver it must be first initialized, using [IC\\_Init\(\)](#) function. Once initialized, it should be de-initialized before initialized again for the same IC module instance, using [IC\\_Deinit\(\)](#). The initialization function does the following operations:

- sets the clock source, clock prescaler
- sets the number of channel input capture are used
- configures in the input capture mode for detection or measurement signal

### Example:

```

/*PAL instance information */
ic_instance_t ic_pall_instance = { IC_INST_TYPE_FTM, 0U };

ic_input_ch_param_t icPalChnConfig[1] =
{
    {
        .hwChannelId      = 0U,
        .inputCaptureMode = IC_DETECT_RISING_EDGE,
        .filterEn         = false,
        .filterValue      = 0U,
        .channelExtension = &ftmChnExtension0,
        .channelCallbackParams = NULL,
        .channelCallbacks  = ic_pall_channel_callBack0
    }
};

channel_extension_ftm_for_ic_t ftmChnExtension0 =
{
    .continuousModeEn = true
};

extension_ftm_for_ic_t ftmExtensionConfig =
{
    .ftmClockSource = FTM_CLOCK_SOURCE_SYSTEMCLK,
    .ftmPrescaler   = FTM_CLOCK_DIVID_BY_1
};

ic_config_t icPall_InitConfig =
{
    .numChannels = 1U,
    .inputChConfig = icPalChnConfig,
    .extension     = &ftmExtensionConfig
};

/* Initialize input capture mode */
IC_Init(&ic_pall_instance, &icPall_InitConfig);

```

### De-initialize a input capture instance

This function will disable the input capture mode. The driver can't be used again until reinitialized. All register are reset to default value and counter is stopped.

#### Example:

```
/* De-initialize input capture mode */  
IC_Deinit(&ic_pall_instance);
```

### Start the channel in the input capture mode

This function will set the channel is in the input capture mode.

#### Example:

```
uint8_t hwChannel = icPall_InitConfig.inputChConfig[0].hwChannelId;  
  
/* Start channel in the input capture mode */  
IC_StartChannel(&ic_pall_instance, hwChannel);
```

### Stop the channel in the input capture mode

This function will set the channel is used in GPIO mode or other peripheral.

#### Example:

```
uint8_t hwChannel = icPall_InitConfig.inputChConfig[0].hwChannelId;  
  
/* Stop channel in the input capture mode */  
IC_StopChannel(&ic_pall_instance, hwChannel);
```

### Get the measured value

The pulse width measurement and the period measurement can be made after the channel input is in the input capture mode. The value is last captured by count. Note that to get true value of measurement at the first of pulse, please use the IC\_GetValueMeasurement function in interrupt.

#### Example:

```
uint16_t retResult = 0U;  
uint8_t hwChannel = icPall_InitConfig.inputChConfig[0].hwChannelId;  
  
/* Get the last captured value */  
retResult = IC_GetMeasurement(&ic_pall_instance, hwChannel);
```

### Enable notifications on the channel

The notification is executed in the callback application with the IC\_EVENT\_MEASUREMENT\_COMPLETE event which indicates that the measurement of input signal is completed.

#### Example:

```
uint8_t hwChannel = icPall_InitConfig.inputChConfig[0].hwChannelId;  
  
/* Enable the notification */  
IC_EnableNotification(&ic_pall_instance, hwChannel);
```

### Disable notifications on the channel

The callback application will be not executed when the notification is disabled.

#### Example:

```
uint8_t hwChannel = icPall_InitConfig.inputChConfig[0].hwChannelId;  
  
/* Disable the notification */  
IC_DisableNotification(&ic_pall_instance, hwChannel);
```

## Important Notes

- Before using the IC PAL driver the module clock must be configured. Refer to Clock Manager for clock configuration.
- The board specific configurations must be done prior to driver after that can call APIs.
- Some features are not available for all IC IPs and incorrect parameters will be handled by DEV\_ASSERT.

## Data Structures

- struct [ic\\_input\\_ch\\_param\\_t](#)  
The configuration structure of input capture parameters for each channel. [More...](#)
- struct [ic\\_config\\_t](#)  
Defines the configuration structures are used in the input capture mode. [More...](#)
- struct [channel\\_extension\\_ftm\\_for\\_ic\\_t](#)  
Defines the extension structure for the channel configuration over FTM. [More...](#)
- struct [extension\\_ftm\\_for\\_ic\\_t](#)  
Defines the extension structure for the input capture mode over FTM. [More...](#)
- struct [ic\\_pal\\_state\\_t](#)  
The internal context structure. [More...](#)

## Enumerations

- enum [ic\\_option\\_mode\\_t](#) {  
[IC\\_DISABLE\\_OPERATION](#) = 0x00U, [IC\\_TIMESTAMP\\_RISING\\_EDGE](#) = 0x01U, [IC\\_TIMESTAMP\\_FALLING\\_EDGE](#) = 0x02U, [IC\\_TIMESTAMP\\_BOTH\\_EDGES](#) = 0x03U,  
[IC\\_MEASURE\\_RISING\\_EDGE\\_PERIOD](#) = 0x04U, [IC\\_MEASURE\\_FALLING\\_EDGE\\_PERIOD](#) = 0x05U, [IC\\_MEASURE\\_PULSE\\_HIGH](#) = 0x06U, [IC\\_MEASURE\\_PULSE\\_LOW](#) = 0x07U }  
The measurement type for input capture mode Implements : [ic\\_option\\_mode\\_t](#) Class.

## Functions

- status\_t [IC\\_Init](#) (const [ic\\_instance\\_t](#) \*const instance, const [ic\\_config\\_t](#) \*configPtr)  
Initializes the input capture mode.
- status\_t [IC\\_Deinit](#) (const [ic\\_instance\\_t](#) \*const instance)  
De-initialize a input capture instance.
- void [IC\\_StartChannel](#) (const [ic\\_instance\\_t](#) \*const instance, uint8\_t channel)  
Start the counter.
- void [IC\\_StopChannel](#) (const [ic\\_instance\\_t](#) \*const instance, uint8\_t channel)  
Stop the counter.
- status\_t [IC\\_SetChannelMode](#) (const [ic\\_instance\\_t](#) \*const instance, uint8\_t channel, [ic\\_option\\_mode\\_t](#) channelMode)  
Get the measured value.
- uint16\_t [IC\\_GetMeasurement](#) (const [ic\\_instance\\_t](#) \*const instance, uint8\_t channel)  
Get the measured value.
- void [IC\\_EnableNotification](#) (const [ic\\_instance\\_t](#) \*const instance, uint8\_t channel)  
Enable channel notifications.
- void [IC\\_DisableNotification](#) (const [ic\\_instance\\_t](#) \*const instance, uint8\_t channel)  
Disable channel notifications.

## 16.68.2 Data Structure Documentation

### 16.68.2.1 struct ic\_input\_ch\_param\_t

The configuration structure of input capture parameters for each channel.

Implements : ic\_input\_ch\_param\_t\_Class

Definition at line 132 of file ic\_pal.h.

#### Data Fields

- uint8\_t [hwChannelId](#)
- [ic\\_option\\_mode\\_t](#) [inputCaptureMode](#)
- bool [filterEn](#)
- uint16\_t [filterValue](#)
- void \* [channelExtension](#)
- void \* [channelCallbackParams](#)
- [ic\\_callback\\_t](#) [channelCallbacks](#)

#### Field Documentation

##### 16.68.2.1.1 void\* channelCallbackParams

The parameter of callback application for channels event

Definition at line 139 of file ic\_pal.h.

##### 16.68.2.1.2 ic\_callback\_t channelCallbacks

The callback function for channels event

Definition at line 140 of file ic\_pal.h.

##### 16.68.2.1.3 void\* channelExtension

The IP specific configuration structure for channel

Definition at line 138 of file ic\_pal.h.

##### 16.68.2.1.4 bool filterEn

Input capture filter state

Definition at line 136 of file ic\_pal.h.

##### 16.68.2.1.5 uint16\_t filterValue

Filter Value

Definition at line 137 of file ic\_pal.h.

##### 16.68.2.1.6 uint8\_t hwChannelId

Physical hardware channel ID

Definition at line 134 of file ic\_pal.h.

##### 16.68.2.1.7 ic\_option\_mode\_t inputCaptureMode

Input capture mode of operation

Definition at line 135 of file ic\_pal.h.

### 16.68.2.2 struct ic\_config\_t

Defines the configuration structures are used in the input capture mode.

Implements : [ic\\_config\\_t\\_Class](#)

Definition at line 148 of file [ic\\_pal.h](#).

#### Data Fields

- [uint8\\_t nNumChannels](#)
- [const ic\\_input\\_ch\\_param\\_t \\* inputChConfig](#)
- [void \\* extension](#)

#### Field Documentation

##### 16.68.2.2.1 void\* extension

IP specific configuration structure

Definition at line 152 of file [ic\\_pal.h](#).

##### 16.68.2.2.2 const ic\_input\_ch\_param\_t\* inputChConfig

Input capture channels configuration

Definition at line 151 of file [ic\\_pal.h](#).

##### 16.68.2.2.3 uint8\_t nNumChannels

Number of input capture channel used

Definition at line 150 of file [ic\\_pal.h](#).

### 16.68.2.3 struct channel\_extension\_ftm\_for\_ic\_t

Defines the extension structure for the channel configuration over FTM.

Part of FTM channel configuration structure Implements : [channel\\_extension\\_ftm\\_for\\_ic\\_t\\_Class](#)

Definition at line 162 of file [ic\\_pal.h](#).

#### Data Fields

- [bool continuousModeEn](#)

#### Field Documentation

##### 16.68.2.3.1 bool continuousModeEn

Continuous measurement state

Definition at line 164 of file [ic\\_pal.h](#).

### 16.68.2.4 struct extension\_ftm\_for\_ic\_t

Defines the extension structure for the input capture mode over FTM.

Part of FTM configuration structure Implements : [extension\\_ftm\\_for\\_ic\\_t\\_Class](#)

Definition at line 173 of file [ic\\_pal.h](#).

#### Data Fields

- [ftm\\_clock\\_source\\_t ftmClockSource](#)
- [ftm\\_clock\\_ps\\_t ftmPrescaler](#)

## Field Documentation

16.68.2.4.1 `ftm_clock_source_t` `ftmClockSource`

Select clock source for FTM

Definition at line 175 of file `ic_pal.h`.

16.68.2.4.2 `ftm_clock_ps_t` `ftmPrescaler`

Register pre-scaler options available in the `ftm_clock_ps_t` enumeration

Definition at line 176 of file `ic_pal.h`.

16.68.2.5 `struct ic_pal_state_t`

The internal context structure.

This structure is used by the driver for its internal logic. It must be provided by the application through the [IC\\_Init\(\)](#) function, then it cannot be freed until the driver is de-initialized using [IC\\_Deinit\(\)](#). The application should make no assumptions about the content of this structure.

Definition at line 248 of file `ic_pal.h`.

## 16.68.3 Enumeration Type Documentation

16.68.3.1 `enum ic_option_mode_t`

The measurement type for input capture mode Implements : `ic_option_mode_t_Class`.

## Enumerator

**`IC_DISABLE_OPERATION`** Have no operation

**`IC_TIMESTAMP_RISING_EDGE`** Rising edge trigger

**`IC_TIMESTAMP_FALLING_EDGE`** Falling edge trigger

**`IC_TIMESTAMP_BOTH_EDGES`** Rising and falling edge trigger

**`IC_MEASURE_RISING_EDGE_PERIOD`** Period measurement between two consecutive rising edges

**`IC_MEASURE_FALLING_EDGE_PERIOD`** Period measurement between two consecutive falling edges

**`IC_MEASURE_PULSE_HIGH`** The time measurement taken for the pulse to remain ON or HIGH state

**`IC_MEASURE_PULSE_LOW`** The time measurement taken for the pulse to remain OFF or LOW state

Definition at line 115 of file `ic_pal.h`.

## 16.68.4 Function Documentation

16.68.4.1 `status_t IC_Deinit ( const ic_instance_t *const instance )`

De-initialize a input capture instance.

This function will disable the input capture mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

## Parameters

<code>in</code>	<code>instance</code>	The pointer to instance number structure.
-----------------	-----------------------	---

**Returns**

Operation status

- STATUS\_SUCCESS: Operation was successful

Definition at line 1043 of file ic\_pal.c.

**16.68.4.2 void IC\_DisableNotification ( const ic\_instance\_t \*const instance, uint8\_t channel )**

Disable channel notifications.

This function disables channel notification.

**Parameters**

in	<i>instance</i>	The pointer to instance number structure.
in	<i>channel</i>	The channel number.

Definition at line 1551 of file ic\_pal.c.

**16.68.4.3 void IC\_EnableNotification ( const ic\_instance\_t \*const instance, uint8\_t channel )**

Enable channel notifications.

This function enables channel notification.

**Parameters**

in	<i>instance</i>	The pointer to instance number structure.
in	<i>channel</i>	The channel number.

Definition at line 1500 of file ic\_pal.c.

**16.68.4.4 uint16\_t IC\_GetMeasurement ( const ic\_instance\_t \*const instance, uint8\_t channel )**

Get the measured value.

This function will get the value of measured signal in ticks.

**Parameters**

in	<i>instance</i>	The pointer to instance number structure.
in	<i>channel</i>	The channel number.

**Returns**

The last value of measured signal in ticks.

Definition at line 1423 of file ic\_pal.c.

**16.68.4.5 status\_t IC\_Init ( const ic\_instance\_t \*const instance, const ic\_config\_t \* configPtr )**

Initializes the input capture mode.

This function will initialize the IC PAL instance, including the other platform specific HW units used together in the input capture mode. This function configures a group of channels in instance to detect or measure the input signal. : If the filter input is enabled on the channel 0,1,2 or 3 over FTM. The filter pre-scaler will be configured to divide by 4. The maximum frequency for the channel input to be detected correctly is FTM input clock divided by 16.

**Parameters**

in	<i>instance</i>	The pointer to instance number structure.
----	-----------------	---



<i>in</i>	<i>configPtr</i>	The pointer to configuration structure.
-----------	------------------	---

**Returns**

Operation status

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 971 of file ic\_pal.c.

**16.68.4.6** `status_t IC_SetChannelMode ( const ic_instance_t *const instance, uint8_t channel, ic_option_mode_t channelMode )`

Get the measured value.

This function will get the value of measured signal in ticks.

**Parameters**

<i>in</i>	<i>instance</i>	The pointer to instance number structure.
<i>in</i>	<i>channel</i>	The channel number.

**Returns**

The last value of measured signal in ticks.

Definition at line 1297 of file ic\_pal.c.

**16.68.4.7** `void IC_StartChannel ( const ic_instance_t *const instance, uint8_t channel )`

Start the counter.

This function start channel counting.

**Parameters**

<i>in</i>	<i>instance</i>	The pointer to instance number structure.
<i>in</i>	<i>channel</i>	The channel number.

Definition at line 1160 of file ic\_pal.c.

**16.68.4.8** `void IC_StopChannel ( const ic_instance_t *const instance, uint8_t channel )`

Stop the counter.

This function stop channel counting.

**Parameters**

<i>in</i>	<i>instance</i>	The pointer to instance number structure.
<i>in</i>	<i>channel</i>	The channel number.

Definition at line 1232 of file ic\_pal.c.

## 16.69 Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL)

### 16.69.1 Detailed Description

Inter Integrated Circuit- Peripheral Abstraction Layer.

The I2C PAL driver allows communication on an I2C bus. It was designed to be portable across all platforms and IPs which support I2C communication.

#### How to integrate I2C in your application

I2C PAL modules need to include a configuration file named `i2c_pal_cfg.h`, which allows the user to specify which IPs are used and how many resources are allocated for each of them (state structures). The following code example shows how to configure one instance for each available I2C IPs.

```
#ifndef I2C_PAL_cfg_H
#define I2C_PAL_cfg_H

/* Define which IP instance will be used in current project */
#define I2C_OVER_LPI2C
#define I2C_OVER_FLEXIO
#define I2C_OVER_I2C
#define I2C_OVER_SWI2C

/* Define the resources necessary for current project */
#define NO_OF_LPI2C_INSTS_FOR_I2C 2
#define NO_OF_FLEXIO_INSTS_FOR_I2C 1
#define NO_OF_I2C_INSTS_FOR_I2C 0
#define NO_OF_SWI2C_INSTS_FOR_I2C 1
#endif /* I2C_PAL_cfg_H */
```

The following table contains the matching between platforms and available IPs

IP/ M↔ CU	S32↔ K116	S32↔ K118	S32↔ K142	S32↔ K144	S32↔ K146	S32↔ K148	M↔ P↔ C5748 G	M↔ P↔ C5746 C	M↔ P↔ C5744 P	S32↔ R274	S32↔ R372	S32↔ K144↔ W	S32↔ K142↔ W
LP↔ I2C	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	NO	NO	NO	NO	NO	Y↔ ES	Y↔ ES
Flex↔ IO	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	NO	NO	NO	NO	NO	Y↔ ES	Y↔ ES
I2C	NO	NO	NO	NO	NO	NO	Y↔ ES	Y↔ ES	NO	Y↔ ES	Y↔ ES	NO	NO
S↔ W↔ I2C	NO	NO	NO	NO	NO	NO	Y↔ ES	Y↔ ES	Y↔ ES	NO	NO	NO	NO

In order to use the I2C driver it must be first initialized in either master or slave mode, using functions [I2C\\_Master↔Init\(\)](#) or [I2C\\_SlaveInit\(\)](#). Once initialized, it cannot be initialized again for the same I2C module instance until it is de-initialized, using [I2C\\_SlaveDeinit\(\)](#) or [I2C\\_MasterDeinit](#). Different I2C module instances can work independently of each other.

In each mode (master/slave) are available two types of transfers: blocking and non-blocking. The functions which initiate blocking transfers will configure the time out for transmission. If time expires the blocking functions will return `STATUS_TIMEOUT` and transmission will be aborted. The blocking functions are: `I2C_MasterSendDataBlocking`, `I2C_MasterReceiveDataBlocking`, `I2C_SlaveSendDataBlocking` and `I2C_SlaveReceiveDataBlocking`.

Slave Mode provides functions for transmitting or receiving data to/from any I2C master. There are two slave operating modes, selected by the field `slaveListening` in the slave configuration structure:

- Slave always listening: the slave interrupt is enabled at initialization time and the slave always listens to the line for a master addressing it. Any events are reported to the application through the callback function provided at initialization time.
- On-demand operation: the slave is commanded to transmit or receive data through the call of [I2C\\_Slave↔SendData\(\)](#) and [I2C\\_SlaveReceiveData\(\)](#) (or their blocking counterparts). The actual moment of the transfer

depends on the I2C master.

The configuration structure includes a special field named extension. It will be used only for I2C transfers over FLEXIO and should contain a pointer to [extension\\_flexio\\_for\\_i2c\\_t](#) structure. The purpose of this structure is to configure which FLEXIO pins are used by the applications and their functionality (SDA and SCL).

#### Important Notes

- The I2C transfers could be done using interrupts and DMA mode.
- FlexIO driver only supports master mode.
- The driver enables the interrupts for the corresponding module, but any interrupt priority setting must be done by the application.
- SWI2C driver supports only master mode.
- For send/receive blocking functions the timeout parameter is unused for SWI2C driver. The driver has a timeout independent of this parameter.
- The baud rate for SWI2C can reach a maximum value of 90Kbps.
- SWI2C baud rate depends on many parameters such as CPU frequency, compiler, optimizations and pull-up resistors. The I2C\_MasterSetBaudrate is considering the maximum baud rate for swi2c device to be 90Kbps and it was tested using gcc compiler with -O1 optimizations, system clock of 200MHz and external pull-up resistors of 2KOhm each.
- Aborting a transfer with the function I2C\_MasterAbortTransferData() can't be done safely due to device limitation; the user must ensure that the address is sent before aborting the transfer.

#### ## Example code ##

```

/* Configure I2C master */
i2c_master_t i2c1_MasterConfig0 =
{
    .slaveAddress      = 10,
    .is10bitAddr       = false,
    .baudRate          = 100000,
    .transferType       = I2C_PAL_USING_INTERRUPTS,
    .operatingMode      = I2C_PAL_STANDARD_MODE,
    .dmaChannel1        = 255,
    .dmaChannel2        = 255,
    .callback           = NULL,
    .callbackParam      = NULL,
    .extension          = NULL
};

/* Configure I2C slave */
i2c_slave_t i2c2_SlaveConfig0 =
{
    .slaveAddress      = 10,
    .is10bitAddr       = false,
    .slaveListening     = true,
    .transferType       = I2C_PAL_USING_INTERRUPTS,
    .dmaChannel         = 255,
    .callback           = i2c2_SlaveCallback0,
    .callbackParam      = NULL
};

i2c_instance_t i2c1_instance = {I2C_INST_TYPE_FLEXIO, 0U};
i2c_instance_t i2c2_instance = {I2C_INST_TYPE_LPI2C, 0U};
i2c_instance_t i2c3_instance = {I2C_INST_TYPE_LPI2C, 1U};

/* Callback for I2C slave */
void i2c2_SlaveCallback0(i2c_slave_event_t slaveEvent, void *userData)
{
    /* Get instance number from userData */
    i2c_instance_t * instance;
    instance = (i2c_instance_t *) userData;

    /* Check the event type:
     * - set RX or TX buffers depending on the master request type
     */
    if (slaveEvent == I2C_SLAVE_EVENT_RX_REQ)
        I2C_SlaveSetRxBuffer(instance, slaveRxBuffer, TRANSFER_SIZE);
}

```

```

    if (slaveEvent == I2C_SLAVE_EVENT_TX_REQ)
        I2C_SlaveSetTxBuffer(instance, slaveTxBuffer, TRANSFER_SIZE);
}

/* Configure FLEXIO pins routing */
extension_flexio_for_i2c_t extension;
extension.sclPin = 1;
extension.sdaPin = 0;
i2c1_MasterConfig0.extension = &extension;

/* Buffers */
uint8_t slaveTxBuffer[16] = {0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF};
uint8_t slaveRxBuffer[16] = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0};
uint8_t masterTxBuffer[16] = {0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF};
uint8_t masterRxBuffer[16] = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0};

/* Initializes I2C master for FlexIO */
I2C_MasterInit(&i2c1_instance, &i2c1_MasterConfig0);

/* Initialize I2C slave instance for LPI2C driver*/
I2C_SlaveInit(&i2c2_instance, &i2c2_SlaveConfig0);

/* FlexIO master sends masterTxBuffer to LPI2C0 configured as slave */
I2C_MasterSendDataBlocking(&i2c1_instance, masterTxBuffer, BUFFER_SIZE, true, 0xFF);

/* Initialize I2C master for LPI2C1 instance */
I2C_MasterInit(&i2c3_instance, &i2c1_MasterConfig0);

/* LPI2C1 master sends data to LPI2C0 configured as slave */
I2C_MasterSendDataBlocking(&i2c3_instance, masterTxBuffer, BUFFER_SIZE, true, 0xFF);

/* De-initialize I2C modules */
I2C_MasterDeinit(&i2c1_instance);
I2C_MasterDeinit(&i2c3_instance);
I2C_SlaveDeinit(&i2c2_instance);

```

## Integration guideline

### Compilation units

The following files need to be compiled in the project:

```
{S32SDK_PATH}\platform\pal\src\i2c\i2c_pal.c
```

### Include path

The following paths need to be added to the include path of the toolchain:

```
{S32SDK_PATH}\platform\pal\inc
i2c_pal_cfg.h path
```

### Compile symbols

No special symbols are required for this component

### Dependencies

i2c swi2c [Low Power Inter-Integrated Circuit \(LPI2C\)](#) flexio\_i2c [Enhanced Direct Memory Access \(eDMA\) OS Interface \(OSIF\)](#) [Clock manager](#) [Interrupt Manager \(Interrupt\)](#)

### Data Structures

- struct [extension\\_flexio\\_for\\_i2c\\_t](#)  
Defines the extension structure for the I2C over FLEXIO Implements : [extension\\_flexio\\_for\\_i2c\\_t](#) Class. [More...](#)
- struct [i2c\\_master\\_t](#)  
Defines the configuration structure for I2C master Implements : [i2c\\_master\\_t](#) Class. [More...](#)

- struct [i2c\\_slave\\_t](#)

Defines the configuration structure for I2C slave Implements : [i2c\\_slave\\_t\\_Class](#). [More...](#)

## Enumerations

- enum [i2c\\_pal\\_transfer\\_type\\_t](#) { [I2C\\_PAL\\_USING\\_DMA](#) = 0U, [I2C\\_PAL\\_USING\\_INTERRUPTS](#) = 1U }

Defines the mechanism to update the rx or tx buffers Implements : [i2c\\_pal\\_transfer\\_type\\_t\\_Class](#).

- enum [i2c\\_operating\\_mode\\_t](#) {  
[I2C\\_PAL\\_STANDARD\\_MODE](#) = 0x0U, [I2C\\_PAL\\_FAST\\_MODE](#) = 0x1U, [I2C\\_PAL\\_FASTPLUS\\_MODE](#) = 0x2U, [I2C\\_PAL\\_HIGHSPEED\\_MODE](#) = 0x3U,  
[I2C\\_PAL\\_ULTRAFast\\_MODE](#) = 0x4U }

Defines the operation mode of the i2c pal Implements : [i2c\\_operating\\_mode\\_t\\_Class](#).

## Functions

- status\_t [I2C\\_MasterInit](#) (const [i2c\\_instance\\_t](#) \*const instance, const [i2c\\_master\\_t](#) \*config)  
*Initializes the I2C module in master mode.*
- status\_t [I2C\\_MasterSendData](#) (const [i2c\\_instance\\_t](#) \*const instance, const uint8\_t \*txBuff, uint32\_t txSize, bool sendStop)  
*Perform a non-blocking send transaction on the I2C bus.*
- status\_t [I2C\\_MasterSendDataBlocking](#) (const [i2c\\_instance\\_t](#) \*const instance, const uint8\_t \*txBuff, uint32\_t txSize, bool sendStop, uint32\_t timeout)  
*Perform a blocking send transaction on the I2C bus.*
- status\_t [I2C\\_MasterReceiveData](#) (const [i2c\\_instance\\_t](#) \*const instance, uint8\_t \*rxBuff, uint32\_t rxSize, bool sendStop)  
*Perform a non-blocking receive transaction on the I2C bus.*
- status\_t [I2C\\_MasterReceiveDataBlocking](#) (const [i2c\\_instance\\_t](#) \*const instance, uint8\_t \*rxBuff, uint32\_t rxSize, bool sendStop, uint32\_t timeout)  
*Perform a blocking receive transaction on the I2C bus.*
- status\_t [I2C\\_MasterSetSlaveAddress](#) (const [i2c\\_instance\\_t](#) \*const instance, const uint16\_t address, const bool is10bitAddr)  
*Set the slave address for the I2C communication.*
- status\_t [I2C\\_MasterDeinit](#) (const [i2c\\_instance\\_t](#) \*const instance)  
*De-initializes the I2C master module.*
- status\_t [I2C\\_GetDefaultMasterConfig](#) ([i2c\\_master\\_t](#) \*config)  
*Gets the default configuration structure for master.*
- status\_t [I2C\\_GetDefaultSlaveConfig](#) ([i2c\\_slave\\_t](#) \*config)  
*Gets the default configuration structure for slave.*
- status\_t [I2C\\_SlaveInit](#) (const [i2c\\_instance\\_t](#) \*const instance, const [i2c\\_slave\\_t](#) \*config)  
*Initializes the I2C module in slave mode.*
- status\_t [I2C\\_SlaveSendData](#) (const [i2c\\_instance\\_t](#) \*const instance, const uint8\_t \*txBuff, uint32\_t txSize)  
*Perform a non-blocking send transaction on the I2C bus.*
- status\_t [I2C\\_SlaveSendDataBlocking](#) (const [i2c\\_instance\\_t](#) \*const instance, const uint8\_t \*txBuff, uint32\_t txSize, uint32\_t timeout)  
*Perform a blocking send transaction on the I2C bus.*
- status\_t [I2C\\_SlaveReceiveData](#) (const [i2c\\_instance\\_t](#) \*const instance, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Perform a non-blocking receive transaction on the I2C bus.*
- status\_t [I2C\\_SlaveReceiveDataBlocking](#) (const [i2c\\_instance\\_t](#) \*const instance, uint8\_t \*rxBuff, uint32\_t rxSize, uint32\_t timeout)  
*Perform a blocking receive transaction on the I2C bus.*
- status\_t [I2C\\_SlaveSetRxBuffer](#) (const [i2c\\_instance\\_t](#) \*const instance, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Provide a buffer for receiving data.*

- status\_t [I2C\\_SlaveSetTxBuffer](#) (const [i2c\\_instance\\_t](#) \*const instance, const uint8\_t \*txBuff, uint32\_t txSize)  
*Provide a buffer for transmitting data.*
- status\_t [I2C\\_SlaveDeinit](#) (const [i2c\\_instance\\_t](#) \*const instance)  
*De-initializes the i2c slave module.*
- status\_t [I2C\\_MasterGetTransferStatus](#) (const [i2c\\_instance\\_t](#) \*const instance, uint32\_t \*bytesRemaining)  
*Return the current status of the I2C master transfer.*
- status\_t [I2C\\_SlaveGetTransferStatus](#) (const [i2c\\_instance\\_t](#) \*const instance, uint32\_t \*bytesRemaining)  
*Return the current status of the I2C slave transfer.*
- status\_t [I2C\\_MasterSetBaudRate](#) (const [i2c\\_instance\\_t](#) \*const instance, const [i2c\\_master\\_t](#) \*config, uint32\_t baudRate)  
*Set the master baud rate for the I2C communication.*
- status\_t [I2C\\_MasterGetBaudRate](#) (const [i2c\\_instance\\_t](#) \*const instance, uint32\_t \*baudRate)  
*Get the master baud rate for the I2C communication.*
- status\_t [I2C\\_MasterAbortTransfer](#) (const [i2c\\_instance\\_t](#) \*const instance)  
*Abort a non-blocking I2C Master transmission or reception.*
- status\_t [I2C\\_SlaveAbortTransfer](#) (const [i2c\\_instance\\_t](#) \*const instance)  
*Abort a non-blocking I2C slave transmission or reception.*

## 16.69.2 Data Structure Documentation

### 16.69.2.1 struct extension\_flexio\_for\_i2c\_t

Defines the extension structure for the I2C over FLEXIO Implements : [extension\\_flexio\\_for\\_i2c\\_t\\_Class](#).

Definition at line 62 of file [i2c\\_pal.h](#).

#### Data Fields

- uint8\_t [sclPin](#)
- uint8\_t [sdaPin](#)

#### Field Documentation

##### 16.69.2.1.1 uint8\_t sclPin

FlexIO pin for SCL

Definition at line 64 of file [i2c\\_pal.h](#).

##### 16.69.2.1.2 uint8\_t sdaPin

FlexIO pin for SDA

Definition at line 65 of file [i2c\\_pal.h](#).

### 16.69.2.2 struct i2c\_master\_t

Defines the configuration structure for I2C master Implements : [i2c\\_master\\_t\\_Class](#).

Definition at line 101 of file [i2c\\_pal.h](#).

#### Data Fields

- uint16\_t [slaveAddress](#)
- bool [is10bitAddr](#)
- uint32\_t [baudRate](#)
- uint8\_t [dmaChannel1](#)
- uint8\_t [dmaChannel2](#)

- [i2c\\_pal\\_transfer\\_type\\_t](#) transferType
- [i2c\\_operating\\_mode\\_t](#) operatingMode
- [i2c\\_master\\_callback\\_t](#) callback
- void \* [callbackParam](#)
- void \* [extension](#)

#### Field Documentation

##### 16.69.2.2.1 uint32\_t baudRate

Baud rate in hertz

Definition at line 105 of file [i2c\\_pal.h](#).

##### 16.69.2.2.2 i2c\_master\_callback\_t callback

User callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if it is not needed

Definition at line 111 of file [i2c\\_pal.h](#).

##### 16.69.2.2.3 void\* callbackParam

Parameter for the callback function

Definition at line 115 of file [i2c\\_pal.h](#).

##### 16.69.2.2.4 uint8\_t dmaChannel1

DMA channel number. Only used in DMA mode

Definition at line 106 of file [i2c\\_pal.h](#).

##### 16.69.2.2.5 uint8\_t dmaChannel2

DMA channel used only by Flexio I2C which needs two DMA channels, one for receiving and one for transmitting.

Definition at line 107 of file [i2c\\_pal.h](#).

##### 16.69.2.2.6 void\* extension

This field will be used to add extra settings to the basic configuration like FlexIO pins

Definition at line 116 of file [i2c\\_pal.h](#).

##### 16.69.2.2.7 bool is10bitAddr

Selects 7-bit or 10-bit slave address

Definition at line 104 of file [i2c\\_pal.h](#).

##### 16.69.2.2.8 i2c\_operating\_mode\_t operatingMode

I2C Operating mode

Definition at line 110 of file [i2c\\_pal.h](#).

##### 16.69.2.2.9 uint16\_t slaveAddress

Slave address, 7-bit or 10-bit

Definition at line 103 of file [i2c\\_pal.h](#).

##### 16.69.2.2.10 i2c\_pal\_transfer\_type\_t transferType

Type of I2C transfer (interrupts or DMA)

Definition at line 109 of file i2c\_pal.h.

#### 16.69.2.3 struct i2c\_slave\_t

Defines the configuration structure for I2C slave Implements : i2c\_slave\_t\_Class.

Definition at line 124 of file i2c\_pal.h.

##### Data Fields

- uint16\_t [slaveAddress](#)
- bool [is10bitAddr](#)
- bool [slaveListening](#)
- [i2c\\_operating\\_mode\\_t](#) [operatingMode](#)
- [i2c\\_pal\\_transfer\\_type\\_t](#) [transferType](#)
- uint8\_t [dmaChannel](#)
- [i2c\\_slave\\_callback\\_t](#) [callback](#)
- void \* [callbackParam](#)

##### Field Documentation

#### 16.69.2.3.1 i2c\_slave\_callback\_t callback

Callback function.

Definition at line 133 of file i2c\_pal.h.

#### 16.69.2.3.2 void\* callbackParam

Parameter for the slave callback function

Definition at line 134 of file i2c\_pal.h.

#### 16.69.2.3.3 uint8\_t dmaChannel

Channel number for DMA channel. If DMA mode is not supported or is not used this field will be ignored.

Definition at line 131 of file i2c\_pal.h.

#### 16.69.2.3.4 bool is10bitAddr

Selects 7-bit or 10-bit slave address

Definition at line 127 of file i2c\_pal.h.

#### 16.69.2.3.5 i2c\_operating\_mode\_t operatingMode

I2C Operating mode

Definition at line 129 of file i2c\_pal.h.

#### 16.69.2.3.6 uint16\_t slaveAddress

Slave address, 7-bit or 10-bit

Definition at line 126 of file i2c\_pal.h.

#### 16.69.2.3.7 bool slaveListening

Slave mode (always listening or on demand only)

Definition at line 128 of file i2c\_pal.h.



16.69.2.3.8 `i2c_pal_transfer_type_t` transferType

Type of the I2C transfer

Definition at line 130 of file `i2c_pal.h`.

## 16.69.3 Enumeration Type Documentation

16.69.3.1 `enum i2c_operating_mode_t`

Defines the operation mode of the i2c pal Implements : `i2c_operating_mode_t_Class`.

## Enumerator

**`I2C_PAL_STANDARD_MODE`** Standard-mode (Sm), bidirectional data transfers up to 100 kbit/s

**`I2C_PAL_FAST_MODE`** Fast-mode (Fm), bidirectional data transfers up to 400 kbit/s

**`I2C_PAL_FASTPLUS_MODE`** Fast-mode Plus (Fm+), bidirectional data transfers up to 1 Mbit/s

**`I2C_PAL_HIGHSPEED_MODE`** High-speed Mode (Hs-mode), bidirectional data transfers up to 3.4 Mbit/s

**`I2C_PAL_ULTRAFAST_MODE`** Ultra Fast Mode (UFm), unidirectional data transfers up to 5 Mbit/s

Definition at line 87 of file `i2c_pal.h`.

16.69.3.2 `enum i2c_pal_transfer_type_t`

Defines the mechanism to update the rx or tx buffers Implements : `i2c_pal_transfer_type_t_Class`.

## Enumerator

**`I2C_PAL_USING_DMA`** The driver will use DMA to perform I2C transfer

**`I2C_PAL_USING_INTERRUPTS`** The driver will use interrupts to perform I2C transfer

Definition at line 52 of file `i2c_pal.h`.

## 16.69.4 Function Documentation

16.69.4.1 `status_t I2C_GetDefaultMasterConfig ( i2c_master_t * config )`

Gets the default configuration structure for master.

The default configuration structure is:

## Parameters

<code>out</code>	<code>config</code>	Pointer to configuration structure
------------------	---------------------	------------------------------------

## Returns

Error or success status returned by API

Definition at line 876 of file `i2c_pal.c`.

16.69.4.2 `status_t I2C_GetDefaultSlaveConfig ( i2c_slave_t * config )`

Gets the default configuration structure for slave.

The default configuration structure is:

**Parameters**

<i>out</i>	<i>config</i>	Pointer to configuration structure
------------	---------------	------------------------------------

**Returns**

Error or success status returned by API

**16.69.4.3 status\_t I2C\_MasterAbortTransfer ( const i2c\_instance\_t \*const instance )**

Abort a non-blocking I2C Master transmission or reception.

**Parameters**

<i>instance</i>	I2C peripheral instance number
-----------------	--------------------------------

**Returns**

Error or success status returned by API

Definition at line 1372 of file i2c\_pal.c.

**16.69.4.4 status\_t I2C\_MasterDeinit ( const i2c\_instance\_t \*const instance )**

De-initializes the I2C master module.

This function de-initialized the I2C master module.

**Parameters**

<i>in</i>	<i>instance</i>	The name of the instance
-----------	-----------------	--------------------------

**Returns**

Error or success status returned by API

Definition at line 635 of file i2c\_pal.c.

**16.69.4.5 status\_t I2C\_MasterGetBaudRate ( const i2c\_instance\_t \*const instance, uint32\_t \* baudRate )**

Get the master baud rate for the I2C communication.

This function returns the master baud rate of the I2C master module.

**Parameters**

<i>instance</i>	I2C peripheral instance number
-----------------	--------------------------------

**Returns**

the baud rate in Hz

Definition at line 821 of file i2c\_pal.c.

**16.69.4.6 status\_t I2C\_MasterGetTransferStatus ( const i2c\_instance\_t \*const instance, uint32\_t \* bytesRemaining )**

Return the current status of the I2C master transfer.

This function can be called during a non-blocking transmission to check the status of the transfer.

## Parameters

<i>instance</i>	I2C peripheral instance number
<i>bytesRemaining</i>	the number of remaining bytes in the active I2C transfer

## Returns

Error or success status returned by API

Definition at line 1267 of file i2c\_pal.c.

#### 16.69.4.7 status\_t I2C\_MasterInit ( const i2c\_instance\_t \*const *instance*, const i2c\_master\_t \* *config* )

Initializes the I2C module in master mode.

This function initializes and enables the requested I2C module in master mode, configuring the bus parameters.

## Parameters

in	<i>instance</i>	The name of the instance
in	<i>config</i>	The configuration structure

## Returns

Error or success status returned by API

Definition at line 210 of file i2c\_pal.c.

#### 16.69.4.8 status\_t I2C\_MasterReceiveData ( const i2c\_instance\_t \*const *instance*, uint8\_t \* *rxBuff*, uint32\_t *rxSize*, bool *sendStop* )

Perform a non-blocking receive transaction on the I2C bus.

This function starts the reception of a block of data from the currently configured slave address and returns immediately. The rest of the reception is handled by the interrupt service routine.

## Parameters

<i>instance</i>	The name of the instance
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred
<i>sendStop</i>	specifies whether or not to generate stop condition after the reception

## Returns

Error or success status returned by API

Definition at line 533 of file i2c\_pal.c.

#### 16.69.4.9 status\_t I2C\_MasterReceiveDataBlocking ( const i2c\_instance\_t \*const *instance*, uint8\_t \* *rxBuff*, uint32\_t *rxSize*, bool *sendStop*, uint32\_t *timeout* )

Perform a blocking receive transaction on the I2C bus.

This function receives a block of data from the currently configured slave address, and only returns when the transmission is complete.

## Parameters

<i>instance</i>	The name of the instance
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred
<i>sendStop</i>	specifies whether or not to generate stop condition after the reception
<i>timeout</i>	timeout for the transfer in milliseconds

#### Returns

Error or success status returned by API

Definition at line 585 of file i2c\_pal.c.

**16.69.4.10** `status_t I2C_MasterSendData ( const i2c_instance_t *const instance, const uint8_t * txBuff, uint32_t txSize, bool sendStop )`

Perform a non-blocking send transaction on the I2C bus.

This function starts the transmission of a block of data to the currently configured slave address and returns immediately. The rest of the transmission is handled by the interrupt service routine.

#### Parameters

<i>instance</i>	The name of the instance
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred
<i>sendStop</i>	specifies whether or not to generate stop condition after the transmission

#### Returns

Error or success status returned by API

Definition at line 422 of file i2c\_pal.c.

**16.69.4.11** `status_t I2C_MasterSendDataBlocking ( const i2c_instance_t *const instance, const uint8_t * txBuff, uint32_t txSize, bool sendStop, uint32_t timeout )`

Perform a blocking send transaction on the I2C bus.

This function sends a block of data to the currently configured slave address, and only returns when the transmission is complete.

#### Parameters

<i>instance</i>	The name of the instance
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred
<i>sendStop</i>	specifies whether or not to generate stop condition after the transmission
<i>timeout</i>	timeout for the transfer in milliseconds

#### Returns

Error or success status returned by API

Definition at line 479 of file i2c\_pal.c.

**16.69.4.12** `status_t I2C_MasterSetBaudRate ( const i2c_instance_t *const instance, const i2c_master_t * config, uint32_t baudRate )`

Set the master baud rate for the I2C communication.

This function sets the master baud rate of the I2C master module.

## Parameters

<i>instance</i>	I2C peripheral instance number
<i>baudRate</i>	the desired baud rate in Hz

Definition at line 742 of file i2c\_pal.c.

**16.69.4.13** `status_t I2C_MasterSetSlaveAddress ( const i2c_instance_t *const instance, const uint16_t address, const bool is10bitAddr )`

Set the slave address for the I2C communication.

This function sets the slave address which will be used for any future transfer initiated by the I2C master.

## Parameters

<i>instance</i>	I2C peripheral instance number
<i>address</i>	slave 7-bit or 10-bit address

Definition at line 689 of file i2c\_pal.c.

**16.69.4.14** `status_t I2C_SlaveAbortTransfer ( const i2c_instance_t *const instance )`

Abort a non-blocking I2C slave transmission or reception.

## Parameters

<i>instance</i>	I2C peripheral instance number
-----------------	--------------------------------

## Returns

Error or success status returned by API

Definition at line 1425 of file i2c\_pal.c.

**16.69.4.15** `status_t I2C_SlaveDeinit ( const i2c_instance_t *const instance )`

De-initializes the i2c slave module.

This function de-initialized the i2c slave module.

## Parameters

<i>in</i>	<i>instance</i>	The name of the instance
-----------	-----------------	--------------------------

## Returns

Error or success status returned by API

Definition at line 1224 of file i2c\_pal.c.

**16.69.4.16** `status_t I2C_SlaveGetTransferStatus ( const i2c_instance_t *const instance, uint32_t * bytesRemaining )`

Return the current status of the I2C slave transfer.

This function can be called during a non-blocking transmission to check the status of the transfer.

## Parameters

<i>instance</i>	I2C peripheral instance number
<i>bytesRemaining</i>	the number of remaining bytes in the active I2C transfer

## Returns

Error or success status returned by API

Definition at line 1324 of file i2c\_pal.c.

#### 16.69.4.17 `status_t I2C_SlaveInit ( const i2c_instance_t *const instance, const i2c_slave_t * config )`

Initializes the I2C module in slave mode.

This function initializes and enables the requested I2C module in slave mode, configuring the bus parameters.

##### Parameters

<code>in</code>	<code>instance</code>	The name of the instance
<code>in</code>	<code>config</code>	The configuration structure

##### Returns

Error or success status returned by API

Definition at line 350 of file `i2c_pal.c`.

#### 16.69.4.18 `status_t I2C_SlaveReceiveData ( const i2c_instance_t *const instance, uint8_t * rxBuff, uint32_t rxSize )`

Perform a non-blocking receive transaction on the I2C bus.

Performs a non-blocking receive transaction on the I2C bus when the slave is not in listening mode (initialized with `slaveListening = false`). It starts the reception and returns immediately. The rest of the reception is handled by the interrupt service routine.

##### Parameters

<code>instance</code>	The name of the instance
<code>rxBuff</code>	pointer to the buffer where to store received data
<code>rxSize</code>	length in bytes of the data to be transferred

##### Returns

Error or success status returned by API

Definition at line 1024 of file `i2c_pal.c`.

#### 16.69.4.19 `status_t I2C_SlaveReceiveDataBlocking ( const i2c_instance_t *const instance, uint8_t * rxBuff, uint32_t rxSize, uint32_t timeout )`

Perform a blocking receive transaction on the I2C bus.

Performs a blocking receive transaction on the I2C bus when the slave is not in listening mode (initialized with `slaveListening = false`). It sets up the reception and then waits for the transfer to complete before returning.

##### Parameters

<code>instance</code>	The name of the instance
<code>rxBuff</code>	pointer to the buffer where to store received data
<code>rxSize</code>	length in bytes of the data to be transferred
<code>timeout</code>	timeout for the transfer in milliseconds

##### Returns

Error or success status returned by API

Definition at line 1074 of file `i2c_pal.c`.

#### 16.69.4.20 `status_t I2C_SlaveSendData ( const i2c_instance_t *const instance, const uint8_t * txBuff, uint32_t txSize )`

Perform a non-blocking send transaction on the I2C bus.

Performs a non-blocking send transaction on the I2C bus when the slave is not in listening mode (initialized with `slaveListening = false`). It starts the transmission and returns immediately. The rest of the transmission is handled by the interrupt service routine.

## Parameters

<i>instance</i>	The name of the instance
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred

## Returns

Error or success status returned by API

Definition at line 924 of file i2c\_pal.c.

**16.69.4.21** `status_t I2C_SlaveSendDataBlocking ( const i2c_instance_t *const instance, const uint8_t * txBuff, uint32_t txSize, uint32_t timeout )`

Perform a blocking send transaction on the I2C bus.

Performs a blocking send transaction on the I2C bus when the slave is not in listening mode (initialized with slave↔ Listening = false). It sets up the transmission and then waits for the transfer to complete before returning.

## Parameters

<i>instance</i>	The name of the instance
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred
<i>timeout</i>	timeout for the transfer in milliseconds

## Returns

Error or success status returned by API

Definition at line 973 of file i2c\_pal.c.

**16.69.4.22** `status_t I2C_SlaveSetRxBuffer ( const i2c_instance_t *const instance, uint8_t * rxBuff, uint32_t rxSize )`

Provide a buffer for receiving data.

This function provides a buffer in which the I2C slave-mode driver can store received data. It can be called for example from the user callback provided at initialization time, when the driver reports events I2C\_SLAVE\_EVENT↔ T\_RX\_REQ or I2C\_SLAVE\_EVENT\_RX\_FULL.

## Parameters

<i>instance</i>	I2C peripheral instance number
<i>rxBuff</i>	pointer to the data to be transferred
<i>rxSize</i>	length in bytes of the data to be transferred

## Returns

Error or success status returned by API

Definition at line 1126 of file i2c\_pal.c.

**16.69.4.23** `status_t I2C_SlaveSetTxBuffer ( const i2c_instance_t *const instance, const uint8_t * txBuff, uint32_t txSize )`

Provide a buffer for transmitting data.

This function provides a buffer from which the I2C slave-mode driver can transmit data. It can be called for example from the user callback provided at initialization time, when the driver reports events I2C\_SLAVE\_EVENT\_TX\_REQ↔ or I2C\_SLAVE\_EVENT\_TX\_EMPTY.

**Parameters**

<i>instance</i>	I2C peripheral instance number
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API

Definition at line 1174 of file i2c\_pal.c.



## 16.70 Interface management

### 16.70.1 Detailed Description

This group contains APIs that help users manage interface(s) in LIN node.

#### Functions

- `I_bool I_ifc_init (I_ifc_handle iii)`  
*Initialize the controller specified by name, i.e. sets up internal functions such as the baud rate. The default schedule set by the `I_ifc_init` call will be the `L_NULL_SCHEDULE` where no frames will be sent and received. This is the first call a user must perform, before using any other interface related LIN API functions. The function returns zero if the initialization was successful and non-zero if failed.*
- `void I_ifc_goto_sleep (I_ifc_handle iii)`  
*Request slave nodes on the cluster connected to the interface to enter bus sleep mode by issuing one go to sleep command. This API is available only for Master nodes.*
- `void I_ifc_wake_up (I_ifc_handle iii)`  
*Transmit the wake up signal.*
- `I_u16 I_ifc_read_status (I_ifc_handle iii)`  
*This function will return the status of the previous communication.*

### 16.70.2 Function Documentation

#### 16.70.2.1 void I\_ifc\_goto\_sleep ( I\_ifc\_handle iii )

Request slave nodes on the cluster connected to the interface to enter bus sleep mode by issuing one go to sleep command. This API is available only for Master nodes.

#### Note

After sending go to sleep command successfully, the master node sets go to sleep flag to 1 and goes to sleep mode. At the end of Go to sleep schedule table, at the end of frame slot of go to sleep command, in `I_sch_tick()` the master node actually switches its active schedule table to Null to stop all communication. To start LIN communication, the master node shall call `I_ifc_wake_up()` to wake up LIN cluster and `I_sch_set()` to activate normal schedule table.

#### Parameters

<code>in</code>	<code>iii</code>	Interface name
-----------------	------------------	----------------

#### Returns

`void`

Definition at line 376 of file `lin_common_api.c`.

#### 16.70.2.2 I\_bool I\_ifc\_init ( I\_ifc\_handle iii )

Initialize the controller specified by name, i.e. sets up internal functions such as the baud rate. The default schedule set by the `I_ifc_init` call will be the `L_NULL_SCHEDULE` where no frames will be sent and received. This is the first call a user must perform, before using any other interface related LIN API functions. The function returns zero if the initialization was successful and non-zero if failed.

**Parameters**

<i>in</i>	<i>iii</i>	Interface name
-----------	------------	----------------

**Returns**

Operation status

- Zero: Initialization was successful.
- Non-zero: Initialization failed.

Definition at line 405 of file lin\_common\_api.c.

**16.70.2.3    `I_u16 I_ifc_read_status ( I_ifc_handle iii )`**

This function will return the status of the previous communication.

**Parameters**

<i>in</i>	<i>iii</i>	Interface name
-----------	------------	----------------

**Returns**

`I_u16`

Definition at line 469 of file lin\_common\_api.c.

**16.70.2.4    `void I_ifc_wake_up ( I_ifc_handle iii )`**

Transmit the wake up signal.

**Parameters**

<i>in</i>	<i>iii</i>	Interface name
-----------	------------	----------------

**Returns**

`void`

Definition at line 455 of file lin\_common\_api.c.

## 16.71 Interrupt Manager (Interrupt)

### 16.71.1 Detailed Description

The S32 SDK Interrupt Manager provides a set of API/services to configure the Interrupt Controller (NVIC).

The Nested-Vectored Interrupt Controller (NVIC) module implements a relocatable vector table supporting many external interrupts, a single non-maskable interrupt (NMI), and priority levels. The NVIC contains the address of the function to execute for a particular handler. The address is fetched via the instruction port allowing parallel register stacking and look-up. The first sixteen entries are allocated to internal sources with the others mapping to MCU-defined interrupts.

#### 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 priority, and other features. Additionally, 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 (see `IRQn_Type`).

This is an example to enable/disable an interrupt for the `ADC0_IRQn`:

```
#include "interrupt_manager.h"

INT_SYS_EnableIRQ(ADC0_IRQn);

INT_SYS_DisableIRQ(ADC0_IRQn);
```

#### Integration guideline

##### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\drivers\src\interrupt\interrupt_manager.c
```

##### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc\
```

##### Compile symbols

No special symbols are required for this component

##### Dependencies

This component does not have any dependencies.

##### Note

1. When the vector table is not in ram (**flash\_vector\_table = 1**):
  - `INT_SYS_InstallHandler` shall check if the function pointer provided as parameter for the new handler is already present in the vector table for the given IRQ number.
  - The user will be required to manually add the correct handlers in the startup files

## Typedefs

- typedef void(\* [isr\\_t](#)) (void)  
*Interrupt handler type.*

## Functions

- void [DefaultISR](#) (void)  
*Default ISR.*

## Interrupt manager APIs

- void [INT\\_SYS\\_InstallHandler](#) (IRQn\_Type irqNumber, const [isr\\_t](#) newHandler, [isr\\_t](#) \*const oldHandler)  
*Installs an interrupt handler routine for a given IRQ number.*
- void [INT\\_SYS\\_EnableIRQ](#) (IRQn\_Type irqNumber)  
*Enables an interrupt for a given IRQ number.*
- 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.*
- void [INT\\_SYS\\_SetPriority](#) (IRQn\_Type irqNumber, uint8\_t priority)  
*Set Interrupt Priority.*
- uint8\_t [INT\\_SYS\\_GetPriority](#) (IRQn\_Type irqNumber)  
*Get Interrupt Priority.*

### 16.71.2 Typedef Documentation

#### 16.71.2.1 typedef void(\* [isr\\_t](#)) (void)

Interrupt handler type.

Definition at line 76 of file interrupt\_manager.h.

### 16.71.3 Function Documentation

#### 16.71.3.1 void [DefaultISR](#) ( void )

Default ISR.

#### 16.71.3.2 void [INT\\_SYS\\_DisableIRQ](#) ( IRQn\_Type *irqNumber* )

Disables an interrupt for a given IRQ number.

This function disables the individual interrupt for a specified IRQ number.

#### Parameters

<i>irqNumber</i>	IRQ number
------------------	------------

Definition at line 219 of file interrupt\_manager.c.

**16.71.3.3 void INT\_SYS\_DisableIRQGlobal ( void )**

Disable system interrupt.

This function disables the global interrupt by calling the core API.

Definition at line 271 of file interrupt\_manager.c.

**16.71.3.4 void INT\_SYS\_EnableIRQ ( IRQn\_Type irqNumber )**

Enables an interrupt for a given IRQ number.

This function enables the individual interrupt for a specified IRQ number.

**Parameters**

<i>irqNumber</i>	IRQ number
------------------	------------

Definition at line 190 of file interrupt\_manager.c.

**16.71.3.5 void INT\_SYS\_EnableIRQGlobal ( void )**

Enables system interrupt.

This function enables the global interrupt by calling the core API.

Definition at line 248 of file interrupt\_manager.c.

**16.71.3.6 uint8\_t INT\_SYS\_GetPriority ( IRQn\_Type irqNumber )**

Get Interrupt Priority.

The function gets the priority of an interrupt.

**Parameters**

<i>irqNumber</i>	Interrupt number.
------------------	-------------------

**Returns**

priority Priority of the interrupt.

Definition at line 354 of file interrupt\_manager.c.

**16.71.3.7 void INT\_SYS\_InstallHandler ( IRQn\_Type irqNumber, const isr\_t newHandler, isr\_t \*const oldHandler )**

Installs an interrupt handler routine for a given IRQ number.

This function lets the application register/replace the interrupt handler for a specified IRQ number. See a chip-specific reference manual for details and the startup\_<SoC>.s file for each chip family to find out the default interrupt handler for each device.

**Note**

This method is applicable only if interrupt vector is copied in RAM.

**Parameters**

<i>irqNumber</i>	IRQ number
<i>newHandler</i>	New interrupt handler routine address pointer
<i>oldHandler</i>	Pointer to a location to store current interrupt handler

Definition at line 114 of file interrupt\_manager.c.

**16.71.3.8 void INT\_SYS\_SetPriority ( IRQn\_Type irqNumber, uint8\_t priority )**

Set Interrupt Priority.

The function sets the priority of an interrupt.

## Parameters

<i>irqNumber</i>	Interrupt number.
<i>priority</i>	Priority to set.

Definition at line 289 of file interrupt\_manager.c.

## 16.72 Interrupt vector numbers for S32K142

This module covers interrupt number allocation.



### 16.73 J2602 Specific API

J2602 protocol is LIN 2.0 based. It contains LIN 2.0's modules to support Signal management, network management, scheduler and J2602 status management. The goal of J2602 is to improve the interoperability and interchangeability of LIN devices within a network by resolving those LIN2.0 requirements that are ambiguous, conflicting, or optional. Moreover, J2602 provides additional requirements that are not present in LIN2.0. For example: fault tolerant, operation, network topology, etc. Different to LIN2.1 protocol, J2602 does not support sporadic and event trigger frames in communication.

## 16.74 J2602 Transport Layer specific API

### 16.74.1 Detailed Description

Contains Transport Layer APIs that only used for J2602 protocol.

#### Modules

- [Node configuration](#)

*This group contains APIs that used for node configuration purpose.*

## 16.75 LIN 2.1 Specific API

### 16.75.1 Detailed Description

LIN 2.1 is extended from in LIN 2.0 specification through diagnostic services and few functions were removed as obsolete.

#### 1. LIN 2.1 is compatible with LIN 2.0:

- A LIN 2.1 master node may handle a LIN 2.0 slave node if the master node also contains all functionality of a LIN 2.0 master node, e.g. obsolete functions like Assign frame Id.
- A LIN 2.1 slave node can be used in a cluster with a LIN 2.0 master node if the LIN 2.1 slave node is pre-configured, i.e. the LIN 2.1 slave node has a valid configuration after reset.

#### 2. Changes between LIN 2.0 and LIN 2.1:

- LIN2.1 enhance the capacity of LIN2.0 on event-triggered frame collision handling and diagnostic services supported. Besides, several features are added to fulfill powerful capacity of LIN network such as configuration service, assign frame ID range configuration, etc.

### Functions

- void [lin\\_collision\\_resolve](#) ([l\\_ifc\\_handle](#) *iii*, [l\\_u8](#) *pid*)  
*Switch to collision resolve table.*
- void [lin\\_update\\_word\\_status\\_lin21](#) ([l\\_ifc\\_handle](#) *iii*, [lin\\_lld\\_event\\_id\\_t](#) *event\_id*)  
*Update node status flags.*
- void [lin\\_update\\_err\\_signal](#) ([l\\_ifc\\_handle](#) *iii*, [l\\_u8](#) *frm\_id*)  
*Update error signal.*
- void [lin\\_make\\_res\\_evnt\\_frame](#) ([l\\_ifc\\_handle](#) *iii*, [l\\_u8](#) *pid*)  
*This function packs signals associated with event trigger frame into buffer.*
- void [lin\\_update\\_rx\\_evnt\\_frame](#) ([l\\_ifc\\_handle](#) *iii*, [l\\_u8](#) *pid*)  
*The function updates the receive flags associated with signals/frames in case receive an event trigger frame.*

### 16.75.2 Function Documentation

#### 16.75.2.1 void [lin\\_collision\\_resolve](#) ( [l\\_ifc\\_handle](#) *iii*, [l\\_u8](#) *pid* )

Switch to collision resolve table.

##### Parameters

<a href="#">in</a>	<a href="#">iii</a>	Interface name
<a href="#">in</a>	<a href="#">pid</a>	PID to process

##### Returns

void

Definition at line 32 of file [lin\\_lin21\\_proto.c](#).

#### 16.75.2.2 void [lin\\_make\\_res\\_evnt\\_frame](#) ( [l\\_ifc\\_handle](#) *iii*, [l\\_u8](#) *pid* )

This function packs signals associated with event trigger frame into buffer.

**Parameters**

in	<i>iii</i>	Interface name
in	<i>pid</i>	PID to process

**Returns**

void

Definition at line 221 of file lin\_lin21\_proto.c.

**16.75.2.3** void lin\_update\_err\_signal ( I\_ifc\_handle *iii*, I\_u8 *frm\_id* )

Update error signal.

**Parameters**

in	<i>iii</i>	Interface name
in	<i>frm_id</i>	Frame index

**Returns**

void

Definition at line 147 of file lin\_lin21\_proto.c.

**16.75.2.4** void lin\_update\_rx\_evt\_frame ( I\_ifc\_handle *iii*, I\_u8 *pid* )

The function updates the receive flags associated with signals/frames in case receive an event trigger frame.

**Parameters**

in	<i>iii</i>	Interface name
in	<i>pid</i>	PID to process

**Returns**

void

Definition at line 184 of file lin\_lin21\_proto.c.

**16.75.2.5** void lin\_update\_word\_status\_lin21 ( I\_ifc\_handle *iii*, lin\_lld\_event\_id\_t *event\_id* )

Update node status flags.

**Parameters**

in	<i>iii</i>	Interface name
in	<i>event_id</i>	Event id

**Returns**

void

Definition at line 67 of file lin\_lin21\_proto.c.

## 16.76 LIN Core API

### 16.76.1 Detailed Description

The LIN core API handles initialization, processing and a signal based interaction between the application and the LIN core. Refer to chapter 7, LIN 2.2A specification.

- Core API layer consists of API functions as defined by the LIN2.1/J2602 specifications.
- Enabling the user to utilize the LIN2.1/J2602 communication within the user application.
- Both the static and dynamic modes for calling the API functions are supported.
- The core API layer interacts with the low level layer and can be called by such upper layers as LIN2.1 TL API, LIN TL J2602 or application for diagnostic implementation.

### Modules

- [Common Core API](#).
- [J2602 Specific API](#)
- [LIN 2.1 Specific API](#)

## 16.77 LIN Driver

### 16.77.1 Detailed Description

This section describes the programming interface of the Peripheral driver for LIN.

### 16.77.2 LIN Driver Overview

The LIN (Local Interconnect Network) Driver is an use-case driven High Level Peripheral Driver. The driver provides users important key features. NXP provides LIN Stack as a middleware software package that is developed on LIN driver. Users also can create their own LIN applications and LIN stack that are compatible with LIN Specification. In this release package, LIN Driver is built on LPUART interface.

### 16.77.3 LIN Driver Device structures

The driver uses instantiations of the [lin\\_state\\_t](#) to maintain the current state of a particular LIN Hardware 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.

#### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\drivers\src\lin\lin_common.c  
${S32SDK_PATH}\platform\drivers\src\lin\lin_driver.c  
${S32SDK_PATH}\platform\drivers\src\lin\lin_irq.c  
${S32SDK_PATH}\platform\drivers\src\lpuart\lin_lpuart_driver.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc\  
${S32SDK_PATH}\platform\drivers\src\lpuart\
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

- [Clock Manager](#)
- [Interrupt Manager \(Interrupt\)](#)
- [OS Interface \(OSIF\)](#)
- [Low Power Universal Asynchronous Receiver-Transmitter \(LPUART\)](#)

## 16.77.4 LIN Driver Initialization

1. To initialize the LIN driver, call the `LIN_DRV_Init()` function and pass the instance number of the relevant LIN hardware interface instance which is LPUART instance in this release.  
For example: to use LPUART0 pass value 0 to the initialization function.

2. Pass a user configuration structure `lin_user_config_t` as shown here:

```
/* LIN Driver configuration structure */
typedef struct {
    uint32_t baudRate;
    bool nodeFunction;
    bool autobaudEnable;
    lin_timer_get_time_interval_t timerGetTimeIntervalCallback;
} lin_user_config_t;
```

3. For LIN, typically the user configures the `lin_user_config_t` instantiation with a baudrate from 1000bps to 20000bps.  
-E.g. 19200 bps `linUserConfig.baudRate = 19200U`.
4. Node function can be MASTER or SLAVE.  
-E.g. `linUserConfig.nodeFunction = MASTER`
5. If users do not want to use Autobaud feature, then just configure `linUserConfig.autobaudEnable = FALSE`.
6. Users shall assign measurement callback function pointer that is `timerGetTimeIntervalCallback`. This function must return time period between two consecutive calls in nano seconds with accuracy at least 0.1 microsecond and if this function is called for the first time, it will start the timer to measure time. When an event (such as detecting a falling edge of a dominant signal while node is in sleep mode) occurs, LIN driver will call `timerGetTimeIntervalCallback` to start time measurement. Then on rising edge of that signal, LIN driver will call `timerGetTimeIntervalCallback` function to get time interval of that dominant signal in nano seconds. If Autobaud feature is enabled, LIN driver uses `timerGetTimeIntervalCallback` to measure two bit time length between two consecutive falling edges of the sync byte in order to evaluate Master's baudrate. Users can implement this function in their applications. -E.g. `linUserConfig.timerGetTimeIntervalCallback = timerGetTimeIntervalCallback0`; This is a code example to set up a FTM0 for LIN Driver:

```
/* Global variables */
uint16_t timerCounterValue[2] = {0u};
uint16_t timerOverflowInterruptCount = 0u;

/* Callback function to get time interval in nano seconds */
uint32_t timerGetTimeIntervalCallback0(uint32_t *ns)
{
    timerCounterValue[1] = (uint16_t) (ftmBase->CNT);
    *ns = ((uint32_t) (timerCounterValue[1] + timerOverflowInterruptCount*65536u - timerCounterValue[0]))*10
        00 / TIMER_1US;
    timerOverflowInterruptCount = 0U;
    timerCounterValue[0] = timerCounterValue[1];
    return 0U;
}
```

7. This is a code example to set up a user LIN Driver configuration instantiation:

```
/* Device instance number as LPUART instance*/
#define LIO (0U)

lin_state_t linState;
lin_user_config_t linUserConfig;
/* Set baudrate 19200 bps */
linUserConfig.baudRate = 19200U;
/* Node is MASTER */
linUserConfig.nodeFunction = MASTER;
/* Disable autobaud feature */
linUserConfig.autobaudEnable = FALSE;
/* Callback function to get time interval in nano seconds */
linUserConfig.timerGetTimeIntervalCallback = (lin_timer_get_time_t)
    timerGetTimeIntervalCallback0;

/* Initialize LIN Hardware interface */
LIN_DRV_Init(LIO, (lin_user_config_t *) &linUserConfig, (
    lin_state_t *) &linState);
```

8. The users are required to initialize a timer for LIN.

E.g. a Flex Timer (FTM). FTM instance should be initialized in Output Compare mode with an interrupt(E.g. FTM0\_Ch0\_Ch1\_IRQHandler) period of about 500 us. Users can choose a different interrupt period that is appropriate to their applications. In timer interrupt handler, users shall call LIN\_DRV\_TimeoutService to handle linCurrentState->timeoutCounter while sending or receiving data.

#### 16.77.5 LIN Data Transfers

The driver implements transmit and receive functions to transfer buffers of data by blocking and non-blocking modes.

The blocking transmit and receive functions include [LIN\\_DRV\\_SendFrameDataBlocking\(\)](#) and the [LIN\\_DRV\\_ReceiveFrameDataBlocking\(\)](#) functions.

The non-blocking (async) transmit and receive functions include the [LIN\\_DRV\\_SendFrameData\(\)](#) and the [LIN\\_DRV\\_ReceiveFrameData\(\)](#) functions.

The [LIN\\_DRV\\_ReceiveFrameData\(\)](#) function is recommended to be called in an interrupt event of receiving PID as implemented in LIN Stack middleware.

The [LIN\\_DRV\\_ReceiveFrameData\(\)](#) function should be called before data is transferring on the LIN bus. The [LIN\\_DRV\\_ReceiveFrameDataBlocking\(\)](#) function should be called before frame is transferring on the LIN bus. Otherwise, some data may be lost.

Master nodes can transmit frame headers in non-blocking mode using [LIN\\_DRV\\_MasterSendHeader\(\)](#).

In all these cases, the functions are interrupt-driven.

#### 16.77.6 Autobaud feature

AUTOBAUD is an extensive feature in LIN Driver which allows a slave node to automatically detect baudrate of LIN bus and adapt its original baudrate to bus value. Auto Baud is applied when the baudrate of the incoming data is unknown. Currently autobaud feature is supported to detect LIN bus baudrates 2400, 4800, 9600, 14400, 19200 bps.

1. If autobaud feature is enabled, at LIN driver initialization slave's baudrate is set to 19200bps. The application should use a timer interrupt in input capture mode of both rising and falling edges(E.g FTM), call [LIN\\_DRV\\_AutoBaudCapture\(uint32\\_t instance\)](#) function to calculate and set Slave's baudrate like Master's baudrate. When receiving a frame header, the slave detect LIN bus's baudrate based on the synchronization byte and adapts its baudrate accordingly. On changing baudrate, the slave set current event ID to LIN\_BAUDRATE\_ADJUSTED and call the callback function. In that callback function users might change the frame data count timeout. Users can look at CallbackHandler() in [lin.c](#) of lin middleware for a reference.

Note: Lin driver should be initiated before initiating a timer interrupt( E.g FTM).

2. Baudrate evaluation process is executed until autobaud successfully. During run-time if LIN bus's baudrate is changed suddenly to a value other than the slave's current baudrate, users shall reset MCU to execute baudrate evaluation process.

#### Note

1. When the vector table is not in ram (**flash\_vector\_table = 1**):
  - INT\_SYS\_InstallHandler shall check if the function pointer provided as parameter for the new handler is already present in the vector table for the given IRQ number.
  - The user will be required to manually add the correct handlers in the startup files

#### Data Structures

- struct [lin\\_user\\_config\\_t](#)



LIN hardware configuration structure Implements : [lin\\_user\\_config\\_t](#) Class. [More...](#)

- struct [lin\\_state\\_t](#)

Runtime state of the LIN driver. [More...](#)

## Macros

- #define [SLAVE](#) 0U
- #define [MASTER](#) 1U
- #define [MAKE\\_PARITY](#) 0U
- #define [CHECK\\_PARITY](#) 1U

## Typedefs

- typedef uint32\_t(\* [lin\\_timer\\_get\\_time\\_interval\\_t](#)) (uint32\_t \*nanoSeconds)  
Callback function to get time interval in nanoseconds Implements : [lin\\_timer\\_get\\_time\\_interval\\_t](#) Class.
- typedef void(\* [lin\\_callback\\_t](#)) (uint32\_t instance, void \*linState)  
LIN Driver callback function type Implements : [lin\\_callback\\_t](#) Class.

## Enumerations

- enum [lin\\_event\\_id\\_t](#) {  
[LIN\\_NO\\_EVENT](#) = 0x00U, [LIN\\_WAKEUP\\_SIGNAL](#) = 0x01U, [LIN\\_BAUDRATE\\_ADJUSTED](#) = 0x02U, [LIN\\_RECV\\_BREAK\\_FIELD\\_OK](#) = 0x03U,  
[LIN\\_SYNC\\_OK](#) = 0x04U, [LIN\\_SYNC\\_ERROR](#) = 0x05U, [LIN\\_PID\\_OK](#) = 0x06U, [LIN\\_PID\\_ERROR](#) = 0x07U,  
[LIN\\_FRAME\\_ERROR](#) = 0x08U, [LIN\\_READBACK\\_ERROR](#) = 0x09U, [LIN\\_CHECKSUM\\_ERROR](#) = 0x0AU,  
[LIN\\_TX\\_COMPLETED](#) = 0x0BU,  
[LIN\\_RX\\_COMPLETED](#) = 0x0CU, [LIN\\_RX\\_OVERRUN](#) = 0x0DU }  
Defines types for an enumerating event related to an Identifier. Implements : [lin\\_event\\_id\\_t](#) Class.
- enum [lin\\_node\\_state\\_t](#) {  
[LIN\\_NODE\\_STATE\\_UNINIT](#) = 0x00U, [LIN\\_NODE\\_STATE\\_SLEEP\\_MODE](#) = 0x01U, [LIN\\_NODE\\_STATE\\_IDLE](#) = 0x02U, [LIN\\_NODE\\_STATE\\_SEND\\_BREAK\\_FIELD](#) = 0x03U,  
[LIN\\_NODE\\_STATE\\_RECV\\_SYNC](#) = 0x04U, [LIN\\_NODE\\_STATE\\_SEND\\_PID](#) = 0x05U, [LIN\\_NODE\\_STATE\\_RECV\\_PID](#) = 0x06U, [LIN\\_NODE\\_STATE\\_RECV\\_DATA](#) = 0x07U,  
[LIN\\_NODE\\_STATE\\_RECV\\_DATA\\_COMPLETED](#) = 0x08U, [LIN\\_NODE\\_STATE\\_SEND\\_DATA](#) = 0x09U, [LIN\\_NODE\\_STATE\\_SEND\\_DATA\\_COMPLETED](#) = 0x0AU }  
Define type for an enumerating LIN Node state. Implements : [lin\\_node\\_state\\_t](#) Class.

## Variables

- [isr\\_t g\\_linLpuartIsrs](#) [LPUART\_INSTANCE\_COUNT]

## LIN DRIVER

- status\_t [LIN\\_DRV\\_Init](#) (uint32\_t instance, [lin\\_user\\_config\\_t](#) \*linUserConfig, [lin\\_state\\_t](#) \*linCurrentState)  
Initializes an instance LIN Hardware Interface for LIN Network.
- void [LIN\\_DRV\\_Deinit](#) (uint32\_t instance)  
Shuts down the LIN Hardware Interface by disabling interrupts and transmitter/receiver.
- void [LIN\\_DRV\\_GetDefaultConfig](#) (bool isMaster, [lin\\_user\\_config\\_t](#) \*linUserConfig)  
Initializes the LIN user configuration structure with default values.
- [lin\\_callback\\_t](#) [LIN\\_DRV\\_InstallCallback](#) (uint32\_t instance, [lin\\_callback\\_t](#) function)  
Installs callback function that is used for [LIN\\_DRV\\_IRQHandler](#).

- status\_t [LIN\\_DRV\\_SendFrameDataBlocking](#) (uint32\_t instance, const uint8\_t \*txBuff, uint8\_t txSize, uint32\_t timeoutMSec)
 

*Sends Frame data out through the LIN Hardware Interface using blocking method. This function will calculate the checksum byte and send it with the frame data. Blocking means that the function does not return until the transmission is complete. This function checks if txSize is in range from 1 to 8. If not, it will return STATUS\_ERROR. This function also returns STATUS\_ERROR if node's current state is in SLEEP mode. This function checks if the isBusBusy is false, if not it will return LIN\_BUS\_BUSY. The function does not return until the transmission is complete. If the transmission is successful, it will return STATUS\_SUCCESS. If not, it will return STATUS\_TIMEOUT.*
- status\_t [LIN\\_DRV\\_SendFrameData](#) (uint32\_t instance, const uint8\_t \*txBuff, uint8\_t txSize)
 

*Sends frame data out through the LIN Hardware Interface using non-blocking method. This enables an a-sync method for transmitting data. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete. This function will calculate the checksum byte and send it with the frame data. The function will return immediately after calling this function. If txSize is equal to 0 or greater than 8 or node's current state is in SLEEP mode then the function will return STATUS\_ERROR. If isBusBusy is currently true then the function will return LIN\_BUS\_BUSY.*
- status\_t [LIN\\_DRV\\_GetTransmitStatus](#) (uint32\_t instance, uint8\_t \*bytesRemaining)
 

*Get status of an on-going non-blocking transmission While sending frame data using non-blocking method, users can use this function to get status of that transmission. The bytesRemaining shows number of bytes that still needed to transmit.*
- status\_t [LIN\\_DRV\\_ReceiveFrameDataBlocking](#) (uint32\_t instance, uint8\_t \*rxBuff, uint8\_t rxSize, uint32\_t timeoutMSec)
 

*Receives frame data through the LIN Hardware Interface using blocking method. This function receives data from LPUART module using blocking method, the function does not return until the receive is complete. The interrupt handler LIN\_LPUART\_DRV\_IRQHandler will check the checksum byte. If the checksum is correct, it will receive the frame data. If the checksum is incorrect, this function will return STATUS\_TIMEOUT and data in rxBuff might be wrong. This function also check if rxSize is in range from 1 to 8. If not, it will return STATUS\_ERROR. This function also returns STATUS\_ERROR if node's current state is in SLEEP mode. This function checks if the isBusBusy is false, if not it will return LIN\_BUS\_BUSY.*
- status\_t [LIN\\_DRV\\_ReceiveFrameData](#) (uint32\_t instance, uint8\_t \*rxBuff, uint8\_t rxSize)
 

*Receives frame data through the LIN Hardware Interface using non-blocking method. This function will check the checksum byte. If the checksum is correct, it will receive it with the frame data. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the reception is complete. The interrupt handler LIN\_LPUART\_DRV\_IRQHandler will check the checksum byte. If the checksum is correct, it will receive the frame data. If the checksum is incorrect, this function will return STATUS\_TIMEOUT and data in rxBuff might be wrong. This function also check if rxSize is in range from 1 to 8. If not, it will return STATUS\_ERROR. This function also returns STATUS\_ERROR if node's current state is in SLEEP mode. This function checks if the isBusBusy is false, if not it will return LIN\_BUS\_BUSY.*
- status\_t [LIN\\_DRV\\_AbortTransferData](#) (uint32\_t instance)
 

*Aborts an on-going non-blocking transmission/reception. While performing a non-blocking transferring data, users can call this function to terminate immediately the transferring.*
- status\_t [LIN\\_DRV\\_GetReceiveStatus](#) (uint32\_t instance, uint8\_t \*bytesRemaining)
 

*Get status of an on-going non-blocking reception. This function returns whether the data reception is complete. When performing non-blocking transmit, the user can call this function to ascertain the state of the current receive progress: in progress (STATUS\_BUSY) or timeout (STATUS\_TIMEOUT) or complete (STATUS\_SUCCESS). In addition, if the reception is still in progress, the user can obtain the number of bytes that still needed to receive.*
- status\_t [LIN\\_DRV\\_GoToSleepMode](#) (uint32\_t instance)
 

*Puts current LIN node to sleep mode This function changes current node state to LIN\_NODE\_STATE\_SLEEP\_MODE.*
- status\_t [LIN\\_DRV\\_GotIdleState](#) (uint32\_t instance)
 

*Puts current LIN node to Idle state This function changes current node state to LIN\_NODE\_STATE\_IDLE.*
- status\_t [LIN\\_DRV\\_SendWakeupSignal](#) (uint32\_t instance)
 

*Sends a wakeup signal through the LIN Hardware Interface.*
- [lin\\_node\\_state\\_t](#) [LIN\\_DRV\\_GetCurrentNodeState](#) (uint32\_t instance)
 

*Get the current LIN node state.*
- void [LIN\\_DRV\\_TimeoutService](#) (uint32\_t instance)

Callback function for Timer Interrupt Handler Users may use (optional, not required) `LIN_DRV_TimeoutService` to check if timeout has occurred during non-blocking frame data transmission and reception. User may initialize a timer (for example FTM) in Output Compare Mode with period of 500 micro seconds (recommended). In timer IRQ handler, call this function.

- void `LIN_DRV_SetTimeoutCounter` (uint32\_t instance, uint32\_t timeoutValue)  
Set Value for Timeout Counter that is used in `LIN_DRV_TimeoutService`.
- status\_t `LIN_DRV_MasterSendHeader` (uint32\_t instance, uint8\_t id)  
Sends frame header out through the LIN Hardware Interface using a non-blocking method. This function sends LIN Break field, sync field then the ID with correct parity. This function checks if the interface is Master, if not, it will return `STATUS_ERROR`. This function checks if id is in range from 0 to 0x3F, if not it will return `STATUS_ERROR`.
- status\_t `LIN_DRV_EnableIRQ` (uint32\_t instance)  
Enables LIN hardware interrupts.
- status\_t `LIN_DRV_DisableIRQ` (uint32\_t instance)  
Disables LIN hardware interrupts.
- void `LIN_DRV_IRQHandler` (uint32\_t instance)  
Interrupt handler for LIN Hardware Interface.
- uint8\_t `LIN_DRV_ProcessParity` (uint8\_t PID, uint8\_t typeAction)  
Makes or checks parity bits. If action is checking parity, the function returns ID value if parity bits are correct or 0xFF if parity bits are incorrect. If action is making parity bits, then from input value of ID, the function returns PID. This is not a public API as it is called by other API functions.
- uint8\_t `LIN_DRV_MakeChecksumByte` (const uint8\_t \*buffer, uint8\_t sizeBuffer, uint8\_t PID)  
Makes the checksum byte for a frame. For PID of identifiers, if PID is 0x3C (ID 0x3C) or 0x7D (ID 0x3D) or 0xFE (ID 0x3E) or 0xBF (ID 0x3F) apply classic checksum and apply enhanced checksum for other PID. In case user want to calculate classic checksum please set PID to zero.
- status\_t `LIN_DRV_AutoBaudCapture` (uint32\_t instance)  
Captures time interval to capture baudrate automatically when enable autobaud feature. This function should only be used in Slave. The timer should be in input capture mode of both rising and falling edges. The timer input capture pin should be externally connected to RXD pin.

## 16.77.7 Data Structure Documentation

### 16.77.7.1 struct lin\_user\_config\_t

LIN hardware configuration structure Implements : `lin_user_config_t_Class`.

Definition at line 70 of file `lin_driver.h`.

#### Data Fields

- uint32\_t `baudRate`
- bool `nodeFunction`
- bool `autobaudEnable`
- `lin_timer_get_time_interval_t` `timerGetTimeIntervalCallback`
- uint8\_t \* `classicPID`
- uint8\_t `numOfClassicPID`

#### Field Documentation

##### 16.77.7.1.1 bool autobaudEnable

Enable Autobaud feature

Definition at line 73 of file `lin_driver.h`.

##### 16.77.7.1.2 uint32\_t baudRate

baudrate of LIN Hardware Interface to configure

Definition at line 71 of file `lin_driver.h`.

### 16.77.7.1.3 uint8\_t\* classicPID

List of PIDs use classic checksum

Definition at line 75 of file lin\_driver.h.

### 16.77.7.1.4 bool nodeFunction

Node function as Master or Slave

Definition at line 72 of file lin\_driver.h.

### 16.77.7.1.5 uint8\_t numOfClassicPID

Number of PIDs use classic checksum

Definition at line 76 of file lin\_driver.h.

### 16.77.7.1.6 lin\_timer\_get\_time\_interval\_t timerGetTimeIntervalCallback

Callback function to get time interval in nanoseconds

Definition at line 74 of file lin\_driver.h.

## 16.77.7.2 struct lin\_state\_t

Runtime state of the LIN driver.

Note that the caller provides memory for the driver state structures during initialization because the driver does not statically allocate memory. Implements : lin\_state\_t\_Class

Definition at line 131 of file lin\_driver.h.

### Data Fields

- const uint8\_t \* txBuff
- uint8\_t \* rxBuff
- uint8\_t cntByte
- volatile uint8\_t txSize
- volatile uint8\_t rxSize
- uint8\_t checkSum
- volatile bool isTxBusy
- volatile bool isRxBusy
- volatile bool isBusBusy
- volatile bool isTxBlocking
- volatile bool isRxBlocking
- lin\_callback\_t Callback
- uint8\_t currentId
- uint8\_t currentPid
- volatile lin\_event\_id\_t currentEventId
- volatile lin\_node\_state\_t currentNodeState
- volatile uint32\_t timeoutCounter
- volatile bool timeoutCounterFlag
- volatile bool baudrateEvalEnable
- volatile uint8\_t fallingEdgeInterruptCount
- uint32\_t linSourceClockFreq
- semaphore\_t txCompleted
- semaphore\_t rxCompleted

## Field Documentation

### 16.77.7.2.1 volatile bool baudrateEvalEnable

Baudrate Evaluation Process Enable

Definition at line 150 of file lin\_driver.h.

### 16.77.7.2.2 lin\_callback\_t Callback

Callback function to invoke after receiving a byte or transmitting a byte.

Definition at line 143 of file lin\_driver.h.

### 16.77.7.2.3 uint8\_t checksum

Checksum byte.

Definition at line 137 of file lin\_driver.h.

### 16.77.7.2.4 uint8\_t cntByte

To count number of bytes already transmitted or received.

Definition at line 134 of file lin\_driver.h.

### 16.77.7.2.5 volatile lin\_event\_id\_t currentEventId

Current ID Event

Definition at line 146 of file lin\_driver.h.

### 16.77.7.2.6 uint8\_t currentId

Current ID

Definition at line 144 of file lin\_driver.h.

### 16.77.7.2.7 volatile lin\_node\_state\_t currentNodeState

Current Node state

Definition at line 147 of file lin\_driver.h.

### 16.77.7.2.8 uint8\_t currentPid

Current PID

Definition at line 145 of file lin\_driver.h.

### 16.77.7.2.9 volatile uint8\_t fallingEdgeInterruptCount

Falling Edge count of a sync byte

Definition at line 151 of file lin\_driver.h.

### 16.77.7.2.10 volatile bool isBusBusy

True if there are data, frame headers being transferred on bus

Definition at line 140 of file lin\_driver.h.

### 16.77.7.2.11 volatile bool isRxBlocking

True if receive is blocking transaction.

Definition at line 142 of file lin\_driver.h.

**16.77.7.2.12 volatile bool isRxBusy**

True if the LIN interface is receiving frame data.

Definition at line 139 of file lin\_driver.h.

**16.77.7.2.13 volatile bool isTxBlocking**

True if transmit is blocking transaction.

Definition at line 141 of file lin\_driver.h.

**16.77.7.2.14 volatile bool isTxBusy**

True if the LIN interface is transmitting frame data.

Definition at line 138 of file lin\_driver.h.

**16.77.7.2.15 uint32\_t linSourceClockFreq**

Frequency of the source clock for LIN

Definition at line 152 of file lin\_driver.h.

**16.77.7.2.16 uint8\_t\* rxBuff**

The buffer of received data.

Definition at line 133 of file lin\_driver.h.

**16.77.7.2.17 semaphore\_t rxCompleted**

Used to wait for LIN interface ISR to complete reception

Definition at line 154 of file lin\_driver.h.

**16.77.7.2.18 volatile uint8\_t rxSize**

The remaining number of bytes to be received.

Definition at line 136 of file lin\_driver.h.

**16.77.7.2.19 volatile uint32\_t timeoutCounter**

Value of the timeout counter

Definition at line 148 of file lin\_driver.h.

**16.77.7.2.20 volatile bool timeoutCounterFlag**

Timeout counter flag

Definition at line 149 of file lin\_driver.h.

**16.77.7.2.21 const uint8\_t\* txBuff**

The buffer of data being sent.

Definition at line 132 of file lin\_driver.h.

**16.77.7.2.22 semaphore\_t txCompleted**

Used to wait for LIN interface ISR to complete transmission.

Definition at line 153 of file lin\_driver.h.

## 16.77.7.2.23 volatile uint8\_t txSize

The remaining number of bytes to be transmitted.

Definition at line 135 of file lin\_driver.h.

## 16.77.8 Macro Definition Documentation

## 16.77.8.1 #define CHECK\_PARITY 1U

Definition at line 50 of file lin\_driver.h.

## 16.77.8.2 #define MAKE\_PARITY 0U

Definition at line 49 of file lin\_driver.h.

## 16.77.8.3 #define MASTER 1U

Definition at line 48 of file lin\_driver.h.

## 16.77.8.4 #define SLAVE 0U

Definition at line 47 of file lin\_driver.h.

## 16.77.9 Typedef Documentation

## 16.77.9.1 typedef void(\* lin\_callback\_t)(uint32\_t instance, void \*linState)

LIN Driver callback function type Implements : lin\_callback\_t\_Class.

Definition at line 122 of file lin\_driver.h.

## 16.77.9.2 typedef uint32\_t(\* lin\_timer\_get\_time\_interval\_t)(uint32\_t \*nanoSeconds)

Callback function to get time interval in nanoseconds Implements : lin\_timer\_get\_time\_interval\_t\_Class.

Definition at line 64 of file lin\_driver.h.

## 16.77.10 Enumeration Type Documentation

## 16.77.10.1 enum lin\_event\_id\_t

Defines types for an enumerating event related to an Identifier. Implements : lin\_event\_id\_t\_Class.

## Enumerator

**LIN\_NO\_EVENT** No event yet

**LIN\_WAKEUP\_SIGNAL** Received a wakeup signal

**LIN\_BAUDRATE\_ADJUSTED** Indicate that baudrate was adjusted to Master's baudrate

**LIN\_RECV\_BREAK\_FIELD\_OK** Indicate that correct Break Field was received

**LIN\_SYNC\_OK** Sync byte is correct

**LIN\_SYNC\_ERROR** Sync byte is incorrect

**LIN\_PID\_OK** PID correct

**LIN\_PID\_ERROR** PID incorrect

**LIN\_FRAME\_ERROR** Framing Error

**LIN\_READBACK\_ERROR** Readback data is incorrect

**LIN\_CHECKSUM\_ERROR** Checksum byte is incorrect

**LIN\_TX\_COMPLETED** Sending data completed

**LIN\_RX\_COMPLETED** Receiving data completed

**LIN\_RX\_OVERRUN** RX overrun flag

Definition at line 83 of file lin\_driver.h.

#### 16.77.10.2 enum lin\_node\_state\_t

Define type for an enumerating LIN Node state. Implements : lin\_node\_state\_t\_Class.

##### Enumerator

**LIN\_NODE\_STATE\_UNINIT** Uninitialized state

**LIN\_NODE\_STATE\_SLEEP\_MODE** Sleep mode state

**LIN\_NODE\_STATE\_IDLE** Idle state

**LIN\_NODE\_STATE\_SEND\_BREAK\_FIELD** Send break field state

**LIN\_NODE\_STATE\_RECV\_SYNC** Receive the synchronization byte state

**LIN\_NODE\_STATE\_SEND\_PID** Send PID state

**LIN\_NODE\_STATE\_RECV\_PID** Receive PID state

**LIN\_NODE\_STATE\_RECV\_DATA** Receive data state

**LIN\_NODE\_STATE\_RECV\_DATA\_COMPLETED** Receive data completed state

**LIN\_NODE\_STATE\_SEND\_DATA** Send data state

**LIN\_NODE\_STATE\_SEND\_DATA\_COMPLETED** Send data completed state

Definition at line 104 of file lin\_driver.h.

#### 16.77.11 Function Documentation

##### 16.77.11.1 status\_t LIN\_DRV\_AbortTransferData ( uint32\_t instance )

Aborts an on-going non-blocking transmission/reception. While performing a non-blocking transferring data, users can call this function to terminate immediately the transferring.

##### Parameters

<i>instance</i>	LIN Hardware Interface instance number
-----------------	--

##### Returns

function always return STATUS\_SUCCESS

Definition at line 266 of file lin\_driver.c.

##### 16.77.11.2 status\_t LIN\_DRV\_AutoBaudCapture ( uint32\_t instance )

Captures time interval to capture baudrate automatically when enable autobaud feature. This function should only be used in Slave. The timer should be in input capture mode of both rising and falling edges. The timer input capture pin should be externally connected to RXD pin.

##### Parameters



<i>instance</i>	LIN Hardware Interface instance number
-----------------	--

**Returns**

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_BUSY: Operation is running.
- STATUS\_ERROR: Operation failed due to break char incorrect, wakeup signal incorrect or calculate baudrate failed.

Definition at line 493 of file lin\_driver.c.

**16.77.11.3 void LIN\_DRV\_Deinit ( uint32\_t *instance* )**

Shuts down the LIN Hardware Interface by disabling interrupts and transmitter/receiver.

**Parameters**

<i>instance</i>	LIN Hardware Interface instance number
-----------------	--

**Returns**

void

Definition at line 80 of file lin\_driver.c.

**16.77.11.4 status\_t LIN\_DRV\_DisableIRQ ( uint32\_t *instance* )**

Disables LIN hardware interrupts.

**Parameters**

<i>instance</i>	LIN Hardware Interface instance number
-----------------	--

**Returns**

function always return STATUS\_SUCCESS.

Definition at line 455 of file lin\_driver.c.

**16.77.11.5 status\_t LIN\_DRV\_EnableIRQ ( uint32\_t *instance* )**

Enables LIN hardware interrupts.

**Parameters**

<i>instance</i>	LIN Hardware Interface instance number.
-----------------	---

**Returns**

function always return STATUS\_SUCCESS.

Definition at line 437 of file lin\_driver.c.

**16.77.11.6 lin\_node\_state\_t LIN\_DRV\_GetCurrentNodeState ( uint32\_t *instance* )**

Get the current LIN node state.

**Parameters**

<i>instance</i>	LIN Hardware Interface instance number
-----------------	--

**Returns**

current LIN node state

Definition at line 363 of file lin\_driver.c.

**16.77.11.7** void LIN\_DRV\_GetDefaultConfig ( bool *isMaster*, lin\_user\_config\_t \* *linUserConfig* )

Initializes the LIN user configuration structure with default values.

This function initializes a configuration structure received from the application with default values. Note: Users shall assign measurement callback function pointer that is timerGetTimeIntervalCallback for linUserConfig. Users can see detail in doxygen.

**Parameters**

in	<i>isMaster</i>	Node function: <ul style="list-style-type: none"> <li>• true if node is MASTER</li> <li>• false if node is SLAVE</li> </ul>
out	<i>linUserConfig</i>	the default configuration

**Returns**

void

Definition at line 95 of file lin\_driver.c.

**16.77.11.8** status\_t LIN\_DRV\_GetReceiveStatus ( uint32\_t *instance*, uint8\_t \* *bytesRemaining* )

Get status of an on-going non-blocking reception. This function returns whether the data reception is complete. When performing non-blocking transmit, the user can call this function to ascertain the state of the current receive progress: in progress (STATUS\_BUSY) or timeout (STATUS\_TIMEOUT) or complete (STATUS\_SUCCESS). In addition, if the reception is still in progress, the user can obtain the number of bytes that still needed to receive.

**Parameters**

<i>instance</i>	LIN Hardware Interface instance number
<i>bytesRemaining</i>	Number of bytes still needed to receive

**Returns**

operation status:

- STATUS\_SUCCESS : The reception is complete.
- STATUS\_TIMEOUT : The reception isn't complete.
- STATUS\_BUSY : The reception is on going

Definition at line 289 of file lin\_driver.c.

**16.77.11.9** status\_t LIN\_DRV\_GetTransmitStatus ( uint32\_t *instance*, uint8\_t \* *bytesRemaining* )

Get status of an on-going non-blocking transmission While sending frame data using non-blocking method, users can use this function to get status of that transmission. The bytesRemaining shows number of bytes that still needed to transmit.

## Parameters

<i>instance</i>	LIN Hardware Interface instance number
<i>bytesRemaining</i>	Number of bytes still needed to transmit

## Returns

operation status:

- STATUS\_SUCCESS : The transmission is successful.
- STATUS\_BUSY : The transmission is sending
- STATUS\_TIMEOUT : Operation failed due to timeout has occurred.

Definition at line 187 of file lin\_driver.c.

## 16.77.11.10 status\_t LIN\_DRV\_GotIdleState ( uint32\_t instance )

Puts current LIN node to Idle state This function changes current node state to LIN\_NODE\_STATE\_IDLE.

## Parameters

<i>instance</i>	LIN Hardware Interface instance number
-----------------	--

## Returns

function always return STATUS\_SUCCESS

Definition at line 327 of file lin\_driver.c.

## 16.77.11.11 status\_t LIN\_DRV\_GoToSleepMode ( uint32\_t instance )

Puts current LIN node to sleep mode This function changes current node state to LIN\_NODE\_STATE\_SLEEP\_↔MODE.

## Parameters

<i>instance</i>	LIN Hardware Interface instance number
-----------------	--

## Returns

function always return STATUS\_SUCCESS

Definition at line 309 of file lin\_driver.c.

## 16.77.11.12 status\_t LIN\_DRV\_Init ( uint32\_t instance, lin\_user\_config\_t \* linUserConfig, lin\_state\_t \* linCurrentState )

Initializes an instance LIN Hardware Interface for LIN Network.

The caller provides memory for the driver state structures during initialization. The user must select the LIN Hardware Interface clock source in the application to initialize the LIN Hardware Interface.

## Parameters

<i>instance</i>	LIN Hardware Interface instance number
<i>linUserConfig</i>	user configuration structure of type <a href="#">lin_user_config_t</a>
<i>linCurrentState</i>	pointer to the LIN Hardware Interface driver state structure

## Returns

operation status:

- STATUS\_SUCCESS : Operation was successful.
- STATUS\_ERROR : Operation failed due to semaphores initialize error.

Definition at line 59 of file lin\_driver.c.

### 16.77.11.13 `lin_callback_t LIN_DRV_InstallCallback ( uint32_t instance, lin_callback_t function )`

Installs callback function that is used for LIN\_DRV\_IRQHandler.

#### Note

After a callback is installed, it bypasses part of the LIN Hardware Interface IRQHandler logic. Therefore, the callback needs to handle the indexes of txBuff and txSize.

#### Parameters

<i>instance</i>	LIN Hardware Interface instance number.
<i>function</i>	the LIN receive callback function.

#### Returns

Former LIN callback function pointer.

Definition at line 111 of file lin\_driver.c.

### 16.77.11.14 `void LIN_DRV_IRQHandler ( uint32_t instance )`

Interrupt handler for LIN Hardware Interface.

#### Parameters

<i>instance</i>	LIN Hardware Interface instance number
-----------------	--

#### Returns

void

Definition at line 475 of file lin\_driver.c.

### 16.77.11.15 `uint8_t LIN_DRV_MakeChecksumByte ( const uint8_t * buffer, uint8_t sizeBuffer, uint8_t PID )`

Makes the checksum byte for a frame. For PID of identifiers, if PID is 0x3C (ID 0x3C) or 0x7D (ID 0x3D) or 0xFE (ID 0x3E) or 0xBF (ID 0x3F) apply classic checksum and apply enhanced checksum for other PID. In case user want to calculate classic checksum please set PID to zero.

#### Parameters

<i>buffer</i>	Pointer to Tx buffer
<i>sizeBuffer</i>	Number of bytes that are contained in the buffer.
<i>PID</i>	Protected Identifier byte.

#### Returns

the checksum byte.

Definition at line 100 of file lin\_common.c.

### 16.77.11.16 `status_t LIN_DRV_MasterSendHeader ( uint32_t instance, uint8_t id )`

Sends frame header out through the LIN Hardware Interface using a non-blocking method. This function sends LIN Break field, sync field then the ID with correct parity. This function checks if the interface is Master, if not, it will return STATUS\_ERROR. This function checks if id is in range from 0 to 0x3F, if not it will return STATUS\_ERROR.

## Parameters

<i>instance</i>	LIN Hardware Interface instance number
<i>id</i>	Frame Identifier

## Returns

operation status:

- STATUS\_SUCCESS : The transmission is successful.
- STATUS\_BUSY : Bus busy flag is true.
- STATUS\_ERROR : The interface isn't Master or id isn't in range from 0 to 0x3F or node's current state is in SLEEP mode.

Definition at line 418 of file lin\_driver.c.

#### 16.77.11.17 uint8\_t LIN\_DRV\_ProcessParity ( uint8\_t PID, uint8\_t typeAction )

Makes or checks parity bits. If action is checking parity, the function returns ID value if parity bits are correct or 0xFF if parity bits are incorrect. If action is making parity bits, then from input value of ID, the function returns PID. This is not a public API as it is called by other API functions.

## Parameters

<i>PID</i>	PID byte in case of checking parity bits or ID byte in case of making parity bits.
<i>typeAction</i>	1 for Checking parity bits, 0 for making parity bits

## Returns

Value has 8 bit:

- 0xFF : Parity bits are incorrect,
- ID : Checking parity bits are correct.
- PID : typeAction is making parity bits.

Definition at line 55 of file lin\_common.c.

#### 16.77.11.18 status\_t LIN\_DRV\_ReceiveFrameData ( uint32\_t instance, uint8\_t \* rxBuff, uint8\_t rxSize )

Receives frame data through the LIN Hardware Interface using non-blocking method. This function will check the checksum byte. If the checksum is correct, it will receive it with the frame data. Non-blocking means that the function returns immediately. The application has to get the receive status to know when the reception is complete. The interrupt handler LIN\_LPUART\_DRV\_IRQHandler will check the checksum byte. If the checksum is correct, it will receive the frame data. If the checksum is incorrect, this function will return STATUS\_TIMEOUT and data in rxBuff might be wrong. This function also check if rxSize is in range from 1 to 8. If not, it will return STATUS\_ERROR. This function also returns STATUS\_ERROR if node's current state is in SLEEP mode. This function checks if the isBusBusy is false, if not it will return LIN\_BUS\_BUSY.

## Note

If users use LIN\_DRV\_TimeoutService in a timer interrupt handler, then before using this function, users have to set timeout counter to an appropriate value by using LIN\_DRV\_SetTimeoutCounter(instance, timeout↵ Value). The timeout value should be big enough to complete the reception. Timeout in real time is (timeout↵ Value) \* (time period that LIN\_DRV\_TimeoutService is called). For example, if LIN\_DRV\_TimeoutService is called in an timer interrupt with period of 500 micro seconds, then timeout in real time is timeoutValue \* 500 micro seconds.

**Parameters**

<i>instance</i>	LIN Hardware Interface instance number
<i>rxBuff</i>	buffer containing 8-bit received data
<i>rxSize</i>	the number of bytes to receive

**Returns**

operation status:

- STATUS\_SUCCESS : The receives frame data is successful.
- STATUS\_TIMEOUT : The checksum is incorrect.
- STATUS\_BUSY : Bus busy flag is true.
- STATUS\_ERROR : Operation failed due is equal to 0 or greater than 8 or node's current state is in SLEEP mode

Definition at line 244 of file lin\_driver.c.

**16.77.11.19** `status_t LIN_DRV_ReceiveFrameDataBlocking ( uint32_t instance, uint8_t * rxBuff, uint8_t rxSize, uint32_t timeoutMSec )`

Receives frame data through the LIN Hardware Interface using blocking method. This function receives data from LPUART module using blocking method, the function does not return until the receive is complete. The interrupt handler LIN\_LPUART\_DRV\_IRQHandler will check the checksum byte. If the checksum is correct, it will receive the frame data. If the checksum is incorrect, this function will return STATUS\_TIMEOUT and data in rxBuff might be wrong. This function also check if rxSize is in range from 1 to 8. If not, it will return STATUS\_ERROR. This function also returns STATUS\_ERROR if node's current state is in SLEEP mode. This function checks if the isBusBusy is false, if not it will return LIN\_BUS\_BUSY.

**Parameters**

<i>instance</i>	LIN Hardware Interface instance number
<i>rxBuff</i>	buffer containing 8-bit received data
<i>rxSize</i>	the number of bytes to receive
<i>timeoutMSec</i>	timeout value in milliseconds

**Returns**

operation status:

- STATUS\_SUCCESS : The receives frame data is successful.
- STATUS\_TIMEOUT : The checksum is incorrect.
- STATUS\_BUSY : Bus busy flag is true.
- STATUS\_ERROR : Operation failed due is equal to 0 or greater than 8 or node's current state is in SLEEP mode

Definition at line 214 of file lin\_driver.c.

**16.77.11.20** `status_t LIN_DRV_SendFrameData ( uint32_t instance, const uint8_t * txBuff, uint8_t txSize )`

Sends frame data out through the LIN Hardware Interface using non-blocking method. This enables an a-sync method for transmitting data. Non-blocking means that the function returns immediately. The application has to get the transmit status to know when the transmit is complete. This function will calculate the checksum byte and send it with the frame data. The function will return immediately after calling this function. If txSize is equal to 0 or greater than 8 or node's current state is in SLEEP mode then the function will return STATUS\_ERROR. If isBusBusy is currently true then the function will return LIN\_BUS\_BUSY.

**Note**

If users use LIN\_DRV\_TimeoutService in a timer interrupt handler, then before using this function, users have to set timeout counter to an appropriate value by using LIN\_DRV\_SetTimeoutCounter(instance, timeout↵Value). The timeout value should be big enough to complete the transmission. Timeout in real time is (timeoutValue) \* (time period that LIN\_DRV\_TimeoutService is called). For example, if LIN\_DRV\_Timeout↵Service is called in an timer interrupt with period of 500 micro seconds, then timeout in real time is timeout↵Value \* 500 micro seconds.

**Parameters**

<i>instance</i>	LIN Hardware Interface instance number
<i>txBuff</i>	source buffer containing 8-bit data chars to send
<i>txSize</i>	the number of bytes to send

**Returns**

operation status:

- STATUS\_SUCCESS : The transmission is successful.
- STATUS\_BUSY : Operation failed due to isBusBusy is currently true.
- STATUS\_ERROR : Operation failed due to txSize is equal to 0 or greater than 8 or node's current state is in SLEEP mode

Definition at line 162 of file lin\_driver.c.

**16.77.11.21** `status_t LIN_DRV_SendFrameDataBlocking ( uint32_t instance, const uint8_t * txBuff, uint8_t txSize, uint32_t timeoutMSec )`

Sends Frame data out through the LIN Hardware Interface using blocking method. This function will calculate the checksum byte and send it with the frame data. Blocking means that the function does not return until the transmission is complete. This function checks if txSize is in range from 1 to 8. If not, it will return STATUS\_ER↵ROR. This function also returns STATUS\_ERROR if node's current state is in SLEEP mode. This function checks if the isBusBusy is false, if not it will return LIN\_BUS\_BUSY. The function does not return until the transmission is complete. If the transmission is successful, it will return STATUS\_SUCCESS. If not, it will return STATUS\_TIME↵OUT.

**Parameters**

<i>instance</i>	LIN Hardware Interface instance number
<i>txBuff</i>	source buffer containing 8-bit data chars to send
<i>txSize</i>	the number of bytes to send
<i>timeoutMSec</i>	timeout value in milliseconds

**Returns**

operation status:

- STATUS\_SUCCESS : The transmission is successful.
- STATUS\_TIMEOUT : The transmission isn't successful.

Definition at line 137 of file lin\_driver.c.

**16.77.11.22** `status_t LIN_DRV_SendWakeupSignal ( uint32_t instance )`

Sends a wakeup signal through the LIN Hardware Interface.

**Parameters**

<i>instance</i>	LIN Hardware Interface instance number
-----------------	--

**Returns**

operation status:

- STATUS\_SUCCESS : Bus busy flag is false.
- STATUS\_BUSY : Bus busy flag is true.

Definition at line 345 of file lin\_driver.c.

**16.77.11.23** void LIN\_DRV\_SetTimeoutCounter ( uint32\_t *instance*, uint32\_t *timeoutValue* )

Set Value for Timeout Counter that is used in LIN\_DRV\_TimeoutService.

**Parameters**

<i>instance</i>	LIN Hardware Interface instance number
<i>timeoutValue</i>	Timeout Value to be set

**Returns**

void

Definition at line 398 of file lin\_driver.c.

**16.77.11.24** void LIN\_DRV\_TimeoutService ( uint32\_t *instance* )

Callback function for Timer Interrupt Handler Users may use (optional, not required) LIN\_DRV\_TimeoutService to check if timeout has occurred during non-blocking frame data transmission and reception. User may initialize a timer (for example FTM) in Output Compare Mode with period of 500 micro seconds (recommended). In timer IRQ handler, call this function.

**Parameters**

<i>instance</i>	LIN Hardware Interface instance number
-----------------	--

**Returns**

void

Definition at line 383 of file lin\_driver.c.

**16.77.12 Variable Documentation**

**16.77.12.1** isr\_t g\_linLpuartIsrs[LPUART\_INSTANCE\_COUNT]

Definition at line 88 of file lin\_irq.c.



## 16.78 LIN Stack

### 16.78.1 Detailed Description

This section covers the functionality of the LIN Stack middleware layer in S32 SDK.

#### Introduction

##### LIN Stack Package Components

LIN Stack is a Middleware package that supports the LIN 1.3, 2.0, 2.1 and above, **LIN2.1** and **J2602** specifications. In LIN Stack, LIN 2.1 covers all LIN 2.1, LIN 2.2 and LIN 2.2A specifications, as the changes following LIN 2.1 are only spelling corrections and clarifications.

- 1. **LIN Stack:**

The layered architecture of the LIN Stack is shown on [Figure 1](#). Such architecture aims maximum reusability of common code base for **LIN2.1** and **J2602** specifications for S32 Freescale automotive MCU portfolio.

The core API layer of **LIN2.1**/**J2602** handles initialization, processing and signal based interaction between applications and LIN Core.

The **LIN2.1** TL (Transport Layer) provides methods for diagnostic services.

The low level layer offers methods for handling signal transmission between user applications and hardware such as interface initialization and deinitialization, frame header sending, response receiving, etc. The low level layer is built on top of **LIN Driver** which is built on top of LPUART HAL layer in the current release.

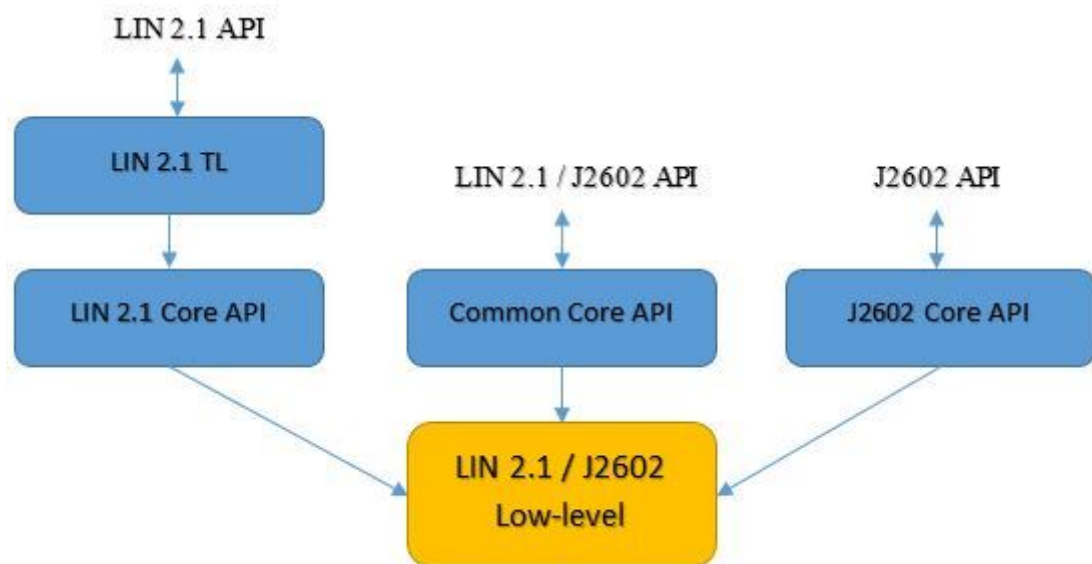


Figure 1. LIN Stack Architecture diagram

#### 2. Node Configuration Tool:

To generate configuration files, users can use the Node Configuration Tool that is LIN Stack PEX component which allows to parse existed LDF files and reflect their contents to LIN Stack component GUI, to create new LDF files, to configure LIN cluster definitions and Node definitions. Using LIN Stack PEX component, users can easily generate the node configuration files (`lin_cfg.h` and `lin_cfg.c`) that are needed for LIN Stack to work properly.

[Figure 2](#). Shows the diagram of configuration data flow.

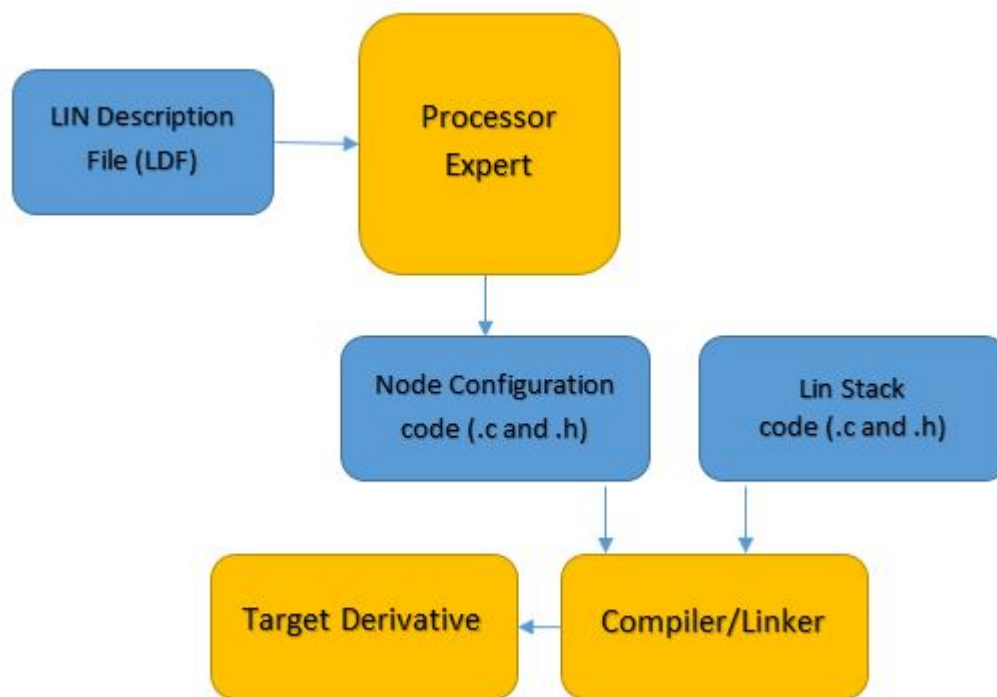


Figure 2. Configuration data

The LDF files describe complete LIN cluster definition including Master/slave mode definition, signals, frames, schedules, timing, etc.

#### Integration guideline

##### Compilation units

The following files need to be compiled in the project:

```

* ${S32SDK_PATH}\middleware\lin\coreapi\lin_common_api.c
* ${S32SDK_PATH}\middleware\lin\coreapi\lin_common_proto.c
* ${S32SDK_PATH}\middleware\lin\coreapi\lin_j2602_proto.c
* ${S32SDK_PATH}\middleware\lin\coreapi\lin_lin21_proto.c
* ${S32SDK_PATH}\middleware\lin\diagnostic\lin_diagnostic_service.c
* ${S32SDK_PATH}\middleware\lin\lowlevel\lin.c
* ${S32SDK_PATH}\middleware\lin\transport\lin_commontl_api.c
* ${S32SDK_PATH}\middleware\lin\transport\lin_commontl_proto.c
*

```

##### Include path

The following paths need to be added to the include path of the toolchain:

```

* ${S32SDK_PATH}\rtos\osif
* ${S32SDK_PATH}\platform\drivers\inc
* ${S32SDK_PATH}\platform\drivers\src\lin
* ${S32SDK_PATH}\platform\drivers\src\lpuart
* ${S32SDK_PATH}\middleware\lin\include
* ${S32SDK_PATH}\middleware\lin\lowlevel
* ${S32SDK_PATH}\middleware\lin\coreapi
* ${S32SDK_PATH}\middleware\lin\transport
*

```

##### Compile symbols

No special symbols are required for this component

##### Dependencies

Clock Manager Pins Driver (PINS) LIN Driver Low Power Universal Asynchronous Receiver-Transmitter (LPUART)

## Modules

- [Diagnostic services](#)

*Diagnostic services defines methods to implement diagnostic data transfer between a master node connected with a diagnostic tester and the slave nodes.*

- [LIN Core API](#)

*The LIN core API handles initialization, processing and a signal based interaction between the application and the LIN core. Refer to chapter 7, LIN 2.2A specification.*

- [Low level API](#)

*Low level layer consists of functions that call LIN driver API.*

- [Transport layer API](#)

*Transport layer stands between the application layer and the core API layer.*

## 16.79 LPI2C Driver

### 16.79.1 Detailed Description

Low Power Inter-Integrated Circuit (LPI2C) Peripheral Driver.

Low Power Inter-Integrated Circuit Driver.

The LPI2C driver allows communication on an I2C bus using the LPI2C module in the S32144K processor.

#### Features

- Interrupt based
- Master or slave operation
- Provides blocking and non-blocking transmit and receive functions
- 7-bit or 10-bit addressing
- Configurable baud rate
- Provides support for all operating modes supported by the hardware
  - Standard-mode (Sm): bidirectional data transfers up to 100 kbit/s
  - Fast-mode (Fm): bidirectional data transfers up to 400 kbit/s

#### Functionality

In order to use the LPI2C driver it must be first initialized in either master or slave mode, using functions [LPI2C\\_DRV\\_MasterInit\(\)](#) or [LPI2C\\_DRV\\_SlaveInit\(\)](#). Once initialized, it cannot be initialized again for the same LPI2C module instance until it is de-initialized, using [LPI2C\\_DRV\\_MasterDeinit\(\)](#) or [LPI2C\\_DRV\\_SlaveDeinit\(\)](#). Different LPI2C module instances can function independently of each other.

#### Master Mode

Master Mode provides functions for transmitting or receiving data to/from any I2C slave. Slave address and baud rate are provided at initialization time through the master configuration structure, but they can be changed at run-time by using [LPI2C\\_DRV\\_MasterSetBaudRate\(\)](#) or [LPI2C\\_DRV\\_MasterSetSlaveAddr\(\)](#). Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency protocol clock for the LPI2C module. The application should call [LPI2C\\_DRV\\_MasterGetBaudRate\(\)](#) after [LPI2C\\_DRV\\_MasterSetBaudRate\(\)](#) to check what baud rate was actually set.

To send or receive data to/from the currently configured slave address, use functions [LPI2C\\_DRV\\_MasterSendData\(\)](#) or [LPI2C\\_DRV\\_MasterReceiveData\(\)](#) (or their blocking counterparts). Parameter `sendStop` can be used to chain multiple transfers with repeated START condition between them, for example when sending a command and then immediately receiving a response. The application should ensure that any send or receive transfer with `sendStop` set to `false` is followed by another transfer, otherwise the LPI2C master will hold the SCL line low indefinitely and block the I2C bus. The last transfer from a chain should always have `sendStop` set to `true`.

Blocking operations will return only when the transfer is completed, either successfully or with error. Non-blocking operations will initiate the transfer and return `STATUS_SUCCESS`, but the module is still busy with the transfer and another transfer can't be initiated until the current transfer is complete. The application can check the status of the current transfer by calling [LPI2C\\_DRV\\_MasterGetTransferStatus\(\)](#). If the transfer is completed, the functions will return either `STATUS_SUCCESS` or an error code, depending on the outcome of the last transfer.

The driver supports any operating mode supported by the module. The operating mode is set together with the baud rate, by [LPI2C\\_DRV\\_MasterSetBaudRate\(\)](#). For High-Speed mode a second baud rate is required, for high-speed communication. Note that due to module limitation (common prescaler setting for normal and fast baud rate) there is a limit on the maximum difference between the two baud rates. [LPI2C\\_DRV\\_MasterGetBaudRate\(\)](#) can be used to check the baud rate setting for both modes.

### Slave Mode

Slave Mode provides functions for transmitting or receiving data to/from any I2C master. There are two slave operating modes, selected by the field `slaveListening` in the slave configuration structure:

- Slave always listening: the slave interrupt is enabled at initialization time and the slave always listens to the line for a master addressing it. Any events are reported to the application through the callback function provided at initialization time. The callback can use [LPI2C\\_DRV\\_SlaveSetRxBuffer\(\)](#) or [LPI2C\\_DRV\\_SlaveSetTxBuffer\(\)](#) to provide the appropriate buffers for transmit or receive, as needed.
- On-demand operation: the slave is commanded to transmit or receive data through the call of [LPI2C\\_DRV\\_SlaveSendData\(\)](#) and [LPI2C\\_DRV\\_SlaveReceiveData\(\)](#) (or their blocking counterparts). The actual moment of the transfer depends on the I2C master. The use of callbacks optional in this case, for example to treat events like `LPI2C_SLAVE_EVENT_TX_EMPTY` or `LPI2C_SLAVE_EVENT_RX_FULL`. Outside the commanded receive / transmit operations the LPI2C interrupts are disabled and the module will not react to master transfer requests.

### Important Notes

- Before using the LPI2C driver in master mode the protocol clock of the module must be configured. Refer to SCG HAL and PCC HAL for clock configuration.
- Before using the LPI2C driver the pins must be routed to the LPI2C module. Refer to PORT HAL for pin routing configuration.
- The driver enables the interrupts for the corresponding LPI2C module, but any interrupt priority setting must be done by the application.
- Fast+, high-speed and ultra-fast mode aren't supported.
- Aborting a master reception is not currently supported due to hardware behavior (the module will continue a started reception even if the FIFO is reset).
- In listening mode, the init function must be called before the master starts the transfer. In non-listening mode, the init function and the appropriate send/receive function must be called before the master starts the transfer.
- Aborting a transfer with the function [LPI2C\\_DRV\\_MasterAbortTransferData\(\)](#) can't be done safely due to device limitation; the user must ensure that the address is sent before aborting the transfer.

### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\drivers\src\lpi2c\lpi2c_irq.c
${S32SDK_PATH}\platform\drivers\src\lpi2c\lpi2c_hw_access.c
${S32SDK_PATH}\platform\drivers\src\lpi2c\lpi2c_driver.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc
${S32SDK_PATH}\platform\drivers\src\lpi2c
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

[Clock Manager OS Interface \(OSIF\)](#) [Interrupt Manager \(Interrupt\)](#) [Enhanced Direct Memory Access \(eDMA\)](#)

## Data Structures

- struct [lpi2c\\_master\\_user\\_config\\_t](#)  
*Defines the example structure. [More...](#)*
- struct [lpi2c\\_slave\\_user\\_config\\_t](#)  
*Slave configuration structure. [More...](#)*
- struct [lpi2c\\_baud\\_rate\\_params\\_t](#)  
*Baud rate structure. [More...](#)*
- struct [lpi2c\\_master\\_state\\_t](#)  
*Master internal context structure. [More...](#)*
- struct [lpi2c\\_slave\\_state\\_t](#)  
*Slave internal context structure. [More...](#)*

## Enumerations

- enum [lpi2c\\_mode\\_t](#) { [LPI2C\\_STANDARD\\_MODE](#) = 0x0U, [LPI2C\\_FAST\\_MODE](#) = 0x1U }  
*I2C operating modes Implements : [lpi2c\\_mode\\_t](#) Class.*
- enum [lpi2c\\_transfer\\_type\\_t](#) { [LPI2C\\_USING\\_DMA](#) = 0, [LPI2C\\_USING\\_INTERRUPTS](#) = 1 }  
*Type of LPI2C transfer (based on interrupts or DMA). Implements : [lpi2c\\_transfer\\_type\\_t](#) Class.*

## LPI2C Driver

- status\_t [LPI2C\\_DRV\\_MasterInit](#) (uint32\_t instance, const [lpi2c\\_master\\_user\\_config\\_t](#) \*userConfigPtr, [lpi2c\\_master\\_state\\_t](#) \*master)  
*Initialize the LPI2C master mode driver.*
- status\_t [LPI2C\\_DRV\\_MasterDeinit](#) (uint32\_t instance)  
*De-initialize the LPI2C master mode driver.*
- void [LPI2C\\_DRV\\_MasterGetBaudRate](#) (uint32\_t instance, [lpi2c\\_baud\\_rate\\_params\\_t](#) \*baudRate)  
*Get the currently configured baud rate.*
- status\_t [LPI2C\\_DRV\\_MasterSetBaudRate](#) (uint32\_t instance, const [lpi2c\\_mode\\_t](#) operatingMode, const [lpi2c\\_baud\\_rate\\_params\\_t](#) baudRate)  
*Set the baud rate for any subsequent I2C communication.*
- void [LPI2C\\_DRV\\_MasterSetSlaveAddr](#) (uint32\_t instance, const uint16\_t address, const bool is10bitAddr)  
*Set the slave address for any subsequent I2C communication.*
- status\_t [LPI2C\\_DRV\\_MasterSendData](#) (uint32\_t instance, const uint8\_t \*txBuff, uint32\_t txSize, bool sendStop)  
*Perform a non-blocking send transaction on the I2C bus.*
- status\_t [LPI2C\\_DRV\\_MasterSendDataBlocking](#) (uint32\_t instance, const uint8\_t \*txBuff, uint32\_t txSize, bool sendStop, uint32\_t timeout)  
*Perform a blocking send transaction on the I2C bus.*
- status\_t [LPI2C\\_DRV\\_MasterAbortTransferData](#) (uint32\_t instance)  
*Abort a non-blocking I2C Master transmission or reception.*
- status\_t [LPI2C\\_DRV\\_MasterReceiveData](#) (uint32\_t instance, uint8\_t \*rxBuff, uint32\_t rxSize, bool sendStop)  
*Perform a non-blocking receive transaction on the I2C bus.*
- status\_t [LPI2C\\_DRV\\_MasterReceiveDataBlocking](#) (uint32\_t instance, uint8\_t \*rxBuff, uint32\_t rxSize, bool sendStop, uint32\_t timeout)  
*Perform a blocking receive transaction on the I2C bus.*
- status\_t [LPI2C\\_DRV\\_MasterGetTransferStatus](#) (uint32\_t instance, uint32\_t \*bytesRemaining)  
*Return the current status of the I2C master transfer.*
- void [LPI2C\\_DRV\\_MasterIRQHandler](#) (uint32\_t instance)  
*Handle master operation when I2C interrupt occurs.*

- status\_t [LPI2C\\_DRV\\_SlaveInit](#) (uint32\_t instance, const [lpi2c\\_slave\\_user\\_config\\_t](#) \*userConfigPtr, [lpi2c\\_slave\\_state\\_t](#) \*slave)
 

*Initialize the I2C slave mode driver.*
- status\_t [LPI2C\\_DRV\\_SlaveDeinit](#) (uint32\_t instance)
 

*De-initialize the I2C slave mode driver.*
- status\_t [LPI2C\\_DRV\\_SlaveSetTxBuffer](#) (uint32\_t instance, const uint8\_t \*txBuff, uint32\_t txSize)
 

*Provide a buffer for transmitting data.*
- status\_t [LPI2C\\_DRV\\_SlaveSetRxBuffer](#) (uint32\_t instance, uint8\_t \*rxBuff, uint32\_t rxSize)
 

*Provide a buffer for receiving data.*
- status\_t [LPI2C\\_DRV\\_SlaveSendData](#) (uint32\_t instance, const uint8\_t \*txBuff, uint32\_t txSize)
 

*Perform a non-blocking send transaction on the I2C bus.*
- status\_t [LPI2C\\_DRV\\_SlaveSendDataBlocking](#) (uint32\_t instance, const uint8\_t \*txBuff, uint32\_t txSize, uint32\_t timeout)
 

*Perform a blocking send transaction on the I2C bus.*
- status\_t [LPI2C\\_DRV\\_SlaveReceiveData](#) (uint32\_t instance, uint8\_t \*rxBuff, uint32\_t rxSize)
 

*Perform a non-blocking receive transaction on the I2C bus.*
- status\_t [LPI2C\\_DRV\\_SlaveReceiveDataBlocking](#) (uint32\_t instance, uint8\_t \*rxBuff, uint32\_t rxSize, uint32\_t timeout)
 

*Perform a blocking receive transaction on the I2C bus.*
- status\_t [LPI2C\\_DRV\\_SlaveGetTransferStatus](#) (uint32\_t instance, uint32\_t \*bytesRemaining)
 

*Return the current status of the I2C slave transfer.*
- status\_t [LPI2C\\_DRV\\_SlaveAbortTransferData](#) (uint32\_t instance)
 

*Abort a non-blocking I2C Master transmission or reception.*
- void [LPI2C\\_DRV\\_SlaveIRQHandler](#) (uint32\_t instance)
 

*Handle slave operation when I2C interrupt occurs.*
- void [LPI2C\\_DRV\\_MasterGetDefaultConfig](#) ([lpi2c\\_master\\_user\\_config\\_t](#) \*config)
 

*Gets the default configuration structure for master.*
- void [LPI2C\\_DRV\\_SlaveGetDefaultConfig](#) ([lpi2c\\_slave\\_user\\_config\\_t](#) \*config)
 

*Gets the default configuration structure for slave.*
- void [LPI2C\\_DRV\\_SetMasterBusIdleTimeout](#) (uint32\_t instance, uint16\_t timeout)
 

*Set bus idle timeout for LPI2C.*

## 16.79.2 Data Structure Documentation

### 16.79.2.1 struct [lpi2c\\_master\\_user\\_config\\_t](#)

Defines the example structure.

This structure is used as an example.

Master configuration structure

This structure is used to provide configuration parameters for the LPI2C master at initialization time. Implements : [lpi2c\\_master\\_user\\_config\\_t\\_Class](#)

Definition at line 111 of file [lpi2c\\_driver.h](#).

#### Data Fields

- uint16\_t [slaveAddress](#)
- bool [is10bitAddr](#)
- [lpi2c\\_mode\\_t](#) [operatingMode](#)
- uint32\_t [baudRate](#)
- [lpi2c\\_transfer\\_type\\_t](#) [transferType](#)
- uint8\_t [dmaChannel](#)
- [i2c\\_master\\_callback\\_t](#) [masterCallback](#)
- void \* [callbackParam](#)

## Field Documentation

### 16.79.2.1.1 `uint32_t baudRate`

The baud rate in hertz to use with current slave device

Definition at line 116 of file `lpi2c_driver.h`.

### 16.79.2.1.2 `void* callbackParam`

Parameter for the master callback function

Definition at line 127 of file `lpi2c_driver.h`.

### 16.79.2.1.3 `uint8_t dmaChannel`

Channel number for DMA channel. If DMA mode isn't used this field will be ignored.

Definition at line 122 of file `lpi2c_driver.h`.

### 16.79.2.1.4 `bool is10bitAddr`

Selects 7-bit or 10-bit slave address

Definition at line 114 of file `lpi2c_driver.h`.

### 16.79.2.1.5 `i2c_master_callback_t masterCallback`

Master callback function. Note that this function will be called from the interrupt service routine at the end of a transfer, so its execution time should be as small as possible. It can be NULL if you want to check manually the status of the transfer.

Definition at line 123 of file `lpi2c_driver.h`.

### 16.79.2.1.6 `lpi2c_mode_t operatingMode`

I2C Operating mode

Definition at line 115 of file `lpi2c_driver.h`.

### 16.79.2.1.7 `uint16_t slaveAddress`

Slave address, 7-bit or 10-bit

Definition at line 113 of file `lpi2c_driver.h`.

### 16.79.2.1.8 `lpi2c_transfer_type_t transferType`

Type of LPI2C transfer

Definition at line 121 of file `lpi2c_driver.h`.

### 16.79.2.2 `struct lpi2c_slave_user_config_t`

Slave configuration structure.

This structure is used to provide configuration parameters for the LPI2C slave at initialization time. Implements : `lpi2c_slave_user_config_t_Class`

Definition at line 136 of file `lpi2c_driver.h`.

## Data Fields

- `uint16_t slaveAddress`
- `bool is10bitAddr`
- `lpi2c_mode_t operatingMode`



- bool [slaveListening](#)
- [lpi2c\\_transfer\\_type\\_t](#) transferType
- [uint8\\_t](#) dmaChannel
- [i2c\\_slave\\_callback\\_t](#) [slaveCallback](#)
- void \* [callbackParam](#)

#### Field Documentation

##### 16.79.2.2.1 void\* callbackParam

Parameter for the slave callback function

Definition at line 149 of file [lpi2c\\_driver.h](#).

##### 16.79.2.2.2 uint8\_t dmaChannel

Channel number for DMA rx channel. If DMA mode isn't used this field will be ignored.

Definition at line 143 of file [lpi2c\\_driver.h](#).

##### 16.79.2.2.3 bool is10bitAddr

Selects 7-bit or 10-bit slave address

Definition at line 139 of file [lpi2c\\_driver.h](#).

##### 16.79.2.2.4 lpi2c\_mode\_t operatingMode

I2C Operating mode

Definition at line 140 of file [lpi2c\\_driver.h](#).

##### 16.79.2.2.5 uint16\_t slaveAddress

Slave address, 7-bit or 10-bit

Definition at line 138 of file [lpi2c\\_driver.h](#).

##### 16.79.2.2.6 i2c\_slave\_callback\_t slaveCallback

Slave callback function. Note that this function will be called from the interrupt service routine, so its execution time should be as small as possible. It can be NULL if the slave is not in listening mode ([slaveListening](#) = false)

Definition at line 144 of file [lpi2c\\_driver.h](#).

##### 16.79.2.2.7 bool slaveListening

Slave mode (always listening or on demand only)

Definition at line 141 of file [lpi2c\\_driver.h](#).

##### 16.79.2.2.8 lpi2c\_transfer\_type\_t transferType

Type of LPI2C transfer

Definition at line 142 of file [lpi2c\\_driver.h](#).

##### 16.79.2.3 struct lpi2c\_baud\_rate\_params\_t

Baud rate structure.

This structure is used for setting or getting the baud rate. Implements : [lpi2c\\_baud\\_rate\\_params\\_t\\_Class](#)

Definition at line 158 of file [lpi2c\\_driver.h](#).

## Data Fields

- uint32\_t [baudRate](#)

## Field Documentation

### 16.79.2.3.1 uint32\_t baudRate

Definition at line 160 of file `lpi2c_driver.h`.

### 16.79.2.4 struct lpi2c\_master\_state\_t

Master internal context structure.

This structure is used by the master-mode driver for its internal logic. It must be provided by the application through the `LPI2C_DRV_MasterInit()` function, then it cannot be freed until the driver is de-initialized using `LPI2C_DRV_M↵MasterDeinit()`. The application should make no assumptions about the content of this structure.

Definition at line 200 of file `lpi2c_driver.h`.

### 16.79.2.5 struct lpi2c\_slave\_state\_t

Slave internal context structure.

This structure is used by the slave-mode driver for its internal logic. It must be provided by the application through the `LPI2C_DRV_SlaveInit()` function, then it cannot be freed until the driver is de-initialized using `LPI2C_DRV_↵SlaveDeinit()`. The application should make no assumptions about the content of this structure.

Definition at line 238 of file `lpi2c_driver.h`.

## 16.79.3 Enumeration Type Documentation

### 16.79.3.1 enum lpi2c\_mode\_t

I2C operating modes Implements : `lpi2c_mode_t_Class`.

## Enumerator

**LPI2C\_STANDARD\_MODE** Standard-mode (Sm), bidirectional data transfers up to 100 kbit/s

**LPI2C\_FAST\_MODE** Fast-mode (Fm), bidirectional data transfers up to 400 kbit/s

Definition at line 71 of file `lpi2c_driver.h`.

### 16.79.3.2 enum lpi2c\_transfer\_type\_t

Type of LPI2C transfer (based on interrupts or DMA). Implements : `lpi2c_transfer_type_t_Class`.

## Enumerator

**LPI2C\_USING\_DMA** The driver will use DMA to perform I2C transfer

**LPI2C\_USING\_INTERRUPTS** The driver will use interrupts to perform I2C transfer

Definition at line 89 of file `lpi2c_driver.h`.

## 16.79.4 Function Documentation

### 16.79.4.1 status\_t LPI2C\_DRV\_MasterAbortTransferData ( uint32\_t instance )

Abort a non-blocking I2C Master transmission or reception.

## Parameters

<i>instance</i>	LPI2C peripheral instance number
-----------------	----------------------------------

## Returns

Error or success status returned by API

Definition at line 1653 of file lpi2c\_driver.c.

#### 16.79.4.2 status\_t LPI2C\_DRV\_MasterDeinit ( uint32\_t *instance* )

De-initialize the LPI2C master mode driver.

This function de-initializes the LPI2C driver in master mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

## Parameters

<i>instance</i>	LPI2C peripheral instance number
-----------------	----------------------------------

## Returns

Error or success status returned by API

Definition at line 1221 of file lpi2c\_driver.c.

#### 16.79.4.3 void LPI2C\_DRV\_MasterGetBaudRate ( uint32\_t *instance*, lpi2c\_baud\_rate\_params\_t \* *baudRate* )

Get the currently configured baud rate.

This function returns the currently configured baud rate.

## Parameters

<i>instance</i>	LPI2C peripheral instance number
<i>baudRate</i>	structure that contains the current baud rate in hertz and the baud rate in hertz for High-speed mode (unused in other modes, can be NULL)

Definition at line 1285 of file lpi2c\_driver.c.

#### 16.79.4.4 void LPI2C\_DRV\_MasterGetDefaultConfig ( lpi2c\_master\_user\_config\_t \* *config* )

Gets the default configuration structure for master.

The default configuration structure is:

## Parameters

<i>config</i>	Pointer to configuration structure
---------------	------------------------------------

Definition at line 1879 of file lpi2c\_driver.c.

#### 16.79.4.5 status\_t LPI2C\_DRV\_MasterGetTransferStatus ( uint32\_t *instance*, uint32\_t \* *bytesRemaining* )

Return the current status of the I2C master transfer.

This function can be called during a non-blocking transmission to check the status of the transfer.

## Parameters

<i>instance</i>	LPI2C peripheral instance number
<i>bytesRemaining</i>	the number of remaining bytes in the active I2C transfer

#### Returns

Error or success status returned by API

Definition at line 1837 of file lpi2c\_driver.c.

**16.79.4.6** `status_t LPI2C_DRV_MasterInit ( uint32_t instance, const lpi2c_master_user_config_t * userConfigPtr, lpi2c_master_state_t * master )`

Initialize the LPI2C master mode driver.

This function initializes the LPI2C driver in master mode.

#### Parameters

<i>instance</i>	LPI2C peripheral instance number
<i>userConfigPtr</i>	Pointer to the LPI2C master user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
<i>master</i>	Pointer to the LPI2C master driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using <a href="#">LPI2C_DRV_MasterDeinit()</a> .

#### Returns

Error or success status returned by API

Definition at line 1136 of file lpi2c\_driver.c.

**16.79.4.7** `void LPI2C_DRV_MasterIRQHandler ( uint32_t instance )`

Handle master operation when I2C interrupt occurs.

This is the interrupt service routine for the LPI2C master mode driver. It handles the rest of the transfer started by one of the send/receive functions.

#### Parameters

<i>instance</i>	LPI2C peripheral instance number
-----------------	----------------------------------

Definition at line 1897 of file lpi2c\_driver.c.

**16.79.4.8** `status_t LPI2C_DRV_MasterReceiveData ( uint32_t instance, uint8_t * rxBuff, uint32_t rxSize, bool sendStop )`

Perform a non-blocking receive transaction on the I2C bus.

This function starts the reception of a block of data from the currently configured slave address and returns immediately. The rest of the reception is handled by the interrupt service routine. Use [LPI2C\\_DRV\\_MasterGetReceiveStatus\(\)](#) to check the progress of the reception.

#### Parameters

<i>instance</i>	LPI2C peripheral instance number
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred

<i>sendStop</i>	specifies whether or not to generate stop condition after the reception
-----------------	---

**Returns**

Error or success status returned by API

Definition at line 1686 of file lpi2c\_driver.c.

**16.79.4.9** `status_t LPI2C_DRV_MasterReceiveDataBlocking ( uint32_t instance, uint8_t * rxBuff, uint32_t rxSize, bool sendStop, uint32_t timeout )`

Perform a blocking receive transaction on the I2C bus.

This function receives a block of data from the currently configured slave address, and only returns when the transmission is complete.

**Parameters**

<i>instance</i>	LPI2C peripheral instance number
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred
<i>sendStop</i>	specifies whether or not to generate stop condition after the reception
<i>timeout</i>	timeout for the transfer in milliseconds

**Returns**

Error or success status returned by API

Definition at line 1782 of file lpi2c\_driver.c.

**16.79.4.10** `status_t LPI2C_DRV_MasterSendData ( uint32_t instance, const uint8_t * txBuff, uint32_t txSize, bool sendStop )`

Perform a non-blocking send transaction on the I2C bus.

This function starts the transmission of a block of data to the currently configured slave address and returns immediately. The rest of the transmission is handled by the interrupt service routine. Use LPI2C\_DRV\_MasterGetSendStatus() to check the progress of the transmission.

**Parameters**

<i>instance</i>	LPI2C peripheral instance number
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred
<i>sendStop</i>	specifies whether or not to generate stop condition after the transmission

**Returns**

Error or success status returned by API

Definition at line 1532 of file lpi2c\_driver.c.

**16.79.4.11** `status_t LPI2C_DRV_MasterSendDataBlocking ( uint32_t instance, const uint8_t * txBuff, uint32_t txSize, bool sendStop, uint32_t timeout )`

Perform a blocking send transaction on the I2C bus.

This function sends a block of data to the currently configured slave address, and only returns when the transmission is complete.

**Parameters**

<i>instance</i>	LPI2C peripheral instance number
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred
<i>sendStop</i>	specifies whether or not to generate stop condition after the transmission
<i>timeout</i>	timeout for the transfer in milliseconds

**Returns**

Error or success status returned by API

Definition at line 1613 of file lpi2c\_driver.c.

**16.79.4.12** `status_t LPI2C_DRV_MasterSetBaudRate ( uint32_t instance, const lpi2c_mode_t operatingMode, const lpi2c_baud_rate_params_t baudRate )`

Set the baud rate for any subsequent I2C communication.

This function sets the baud rate (SCL frequency) for the I2C master. It can also change the operating mode. If the operating mode is High-Speed, a second baud rate must be provided for high-speed communication. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences, for example if requesting a high baud rate while using a low-frequency protocol clock for the LPI2C module. The application should call [LPI2C\\_DRV\\_MasterGetBaudRate\(\)](#) after [LPI2C\\_DRV\\_MasterSetBaudRate\(\)](#) to check what baud rate was actually set.

**Parameters**

<i>instance</i>	LPI2C peripheral instance number
<i>operatingMode</i>	I2C operating mode
<i>baudRate</i>	structure that contains the baud rate in hertz to use by current slave device and also the baud rate in hertz for High-speed mode (unused in other modes)

**Returns**

Error or success status returned by API

Definition at line 1337 of file lpi2c\_driver.c.

**16.79.4.13** `void LPI2C_DRV_MasterSetSlaveAddr ( uint32_t instance, const uint16_t address, const bool is10bitAddr )`

Set the slave address for any subsequent I2C communication.

This function sets the slave address which will be used for any future transfer initiated by the LPI2C master.

**Parameters**

<i>instance</i>	LPI2C peripheral instance number
<i>address</i>	slave address, 7-bit or 10-bit
<i>is10bitAddr</i>	specifies if provided address is 10-bit

Definition at line 1511 of file lpi2c\_driver.c.

**16.79.4.14** `void LPI2C_DRV_SetMasterBusIdleTimeout ( uint32_t instance, uint16_t timeout )`

Set bus idle timeout for LPI2C.

This function sets time out for bus idle for Master.If both SCL and SDA are high for longer than Timeout cycles, then the I2C bus is assumed to be idle and the master can generate a START condition

## Parameters

<i>baseAddr</i>	base address of the LPI2C module
<i>timeout</i>	bus idle timeout period in clock cycle. Zero means no bus idle timeout

Definition at line 1253 of file lpi2c\_driver.c.

**16.79.4.15** `status_t LPI2C_DRV_SlaveAbortTransferData ( uint32_t instance )`

Abort a non-blocking I2C Master transmission or reception.

## Parameters

<i>instance</i>	LPI2C peripheral instance number
-----------------	----------------------------------

## Returns

Error or success status returned by API

Definition at line 2524 of file lpi2c\_driver.c.

**16.79.4.16** `status_t LPI2C_DRV_SlaveDeinit ( uint32_t instance )`

De-initialize the I2C slave mode driver.

This function de-initializes the LPI2C driver in slave mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

## Parameters

<i>instance</i>	LPI2C peripheral instance number
-----------------	----------------------------------

## Returns

Error or success status returned by API

Definition at line 2153 of file lpi2c\_driver.c.

**16.79.4.17** `void LPI2C_DRV_SlaveGetDefaultConfig ( lpi2c_slave_user_config_t * config )`

Gets the default configuration structure for slave.

The default configuration structure is:

## Parameters

<i>config</i>	Pointer to configuration structure
---------------	------------------------------------

Definition at line 2551 of file lpi2c\_driver.c.

**16.79.4.18** `status_t LPI2C_DRV_SlaveGetTransferStatus ( uint32_t instance, uint32_t * bytesRemaining )`

Return the current status of the I2C slave transfer.

This function can be called during a non-blocking transmission to check the status of the transfer.

## Parameters

<i>instance</i>	LPI2C peripheral instance number
<i>bytesRemaining</i>	the number of remaining bytes in the active I2C transfer

## Returns

Error or success status returned by API

Definition at line 2485 of file lpi2c\_driver.c.

16.79.4.19 `status_t LPI2C_DRV_SlaveInit ( uint32_t instance, const lpi2c_slave_user_config_t * userConfigPtr,  
lpi2c_slave_state_t * slave )`

Initialize the I2C slave mode driver.



## Parameters

<i>instance</i>	LPI2C peripheral instance number
<i>userConfigPtr</i>	Pointer to the LPI2C slave user configuration structure. The function reads configuration data from this structure and initializes the driver accordingly. The application may free this structure after the function returns.
<i>slave</i>	Pointer to the LPI2C slave driver context structure. The driver uses this memory area for its internal logic. The application must make no assumptions about the content of this structure, and must not free this memory until the driver is de-initialized using <a href="#">LPI2C_DRV_Slave↵Deinit()</a> .

## Returns

Error or success status returned by API

Definition at line 2032 of file lpi2c\_driver.c.

**16.79.4.20** void LPI2C\_DRV\_SlaveIRQHandler ( uint32\_t *instance* )

Handle slave operation when I2C interrupt occurs.

This is the interrupt service routine for the LPI2C slave mode driver. It handles any transfer initiated by an I2C master and notifies the application via the provided callback when relevant events occur.

## Parameters

<i>instance</i>	LPI2C peripheral instance number
-----------------	----------------------------------

Definition at line 2606 of file lpi2c\_driver.c.

**16.79.4.21** status\_t LPI2C\_DRV\_SlaveReceiveData ( uint32\_t *instance*, uint8\_t \* *rxBuff*, uint32\_t *rxSize* )

Perform a non-blocking receive transaction on the I2C bus.

Performs a non-blocking receive transaction on the I2C bus when the slave is not in listening mode (initialized with slaveListening = false). It starts the reception and returns immediately. The rest of the reception is handled by the interrupt service routine. Use LPI2C\_DRV\_SlaveGetReceiveStatus() to check the progress of the reception.

## Parameters

<i>instance</i>	LPI2C peripheral instance number
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred

## Returns

Error or success status returned by API

Definition at line 2370 of file lpi2c\_driver.c.

**16.79.4.22** status\_t LPI2C\_DRV\_SlaveReceiveDataBlocking ( uint32\_t *instance*, uint8\_t \* *rxBuff*, uint32\_t *rxSize*, uint32\_t *timeout* )

Perform a blocking receive transaction on the I2C bus.

Performs a blocking receive transaction on the I2C bus when the slave is not in listening mode (initialized with slaveListening = false). It sets up the reception and then waits for the transfer to complete before returning.

## Parameters

<i>instance</i>	LPI2C peripheral instance number
<i>rxBuff</i>	pointer to the buffer where to store received data
<i>rxSize</i>	length in bytes of the data to be transferred
<i>timeout</i>	timeout for the transfer in milliseconds

#### Returns

Error or success status returned by API

Definition at line 2443 of file lpi2c\_driver.c.

**16.79.4.23** `status_t LPI2C_DRV_SlaveSendData ( uint32_t instance, const uint8_t * txBuff, uint32_t txSize )`

Perform a non-blocking send transaction on the I2C bus.

Performs a non-blocking send transaction on the I2C bus when the slave is not in listening mode (initialized with slaveListening = false). It starts the transmission and returns immediately. The rest of the transmission is handled by the interrupt service routine. Use LPI2C\_DRV\_SlaveGetTransmitStatus() to check the progress of the transmission.

#### Parameters

<i>instance</i>	LPI2C peripheral instance number
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred

#### Returns

Error or success status returned by API

Definition at line 2246 of file lpi2c\_driver.c.

**16.79.4.24** `status_t LPI2C_DRV_SlaveSendDataBlocking ( uint32_t instance, const uint8_t * txBuff, uint32_t txSize, uint32_t timeout )`

Perform a blocking send transaction on the I2C bus.

Performs a blocking send transaction on the I2C bus when the slave is not in listening mode (initialized with slaveListening = false). It sets up the transmission and then waits for the transfer to complete before returning.

#### Parameters

<i>instance</i>	LPI2C peripheral instance number
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred
<i>timeout</i>	timeout for the transfer in milliseconds

#### Returns

Error or success status returned by API

Definition at line 2332 of file lpi2c\_driver.c.

**16.79.4.25** `status_t LPI2C_DRV_SlaveSetRxBuffer ( uint32_t instance, uint8_t * rxBuff, uint32_t rxSize )`

Provide a buffer for receiving data.

This function provides a buffer in which the LPI2C slave-mode driver can store received data. It can be called for example from the user callback provided at initialization time, when the driver reports events LPI2C\_SLAVE\_EVENT\_RX\_REQ or LPI2C\_SLAVE\_EVENT\_RX\_FULL.

**Parameters**

<i>instance</i>	LPI2C peripheral instance number
<i>rxBuff</i>	pointer to the data to be transferred
<i>rxSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API

Definition at line 2219 of file lpi2c\_driver.c.

**16.79.4.26** `status_t LPI2C_DRV_SlaveSetTxBuffer ( uint32_t instance, const uint8_t * txBuff, uint32_t txSize )`

Provide a buffer for transmitting data.

This function provides a buffer from which the LPI2C slave-mode driver can transmit data. It can be called for example from the user callback provided at initialization time, when the driver reports events LPI2C\_SLAVE\_EVENT\_TX\_REQ or LPI2C\_SLAVE\_EVENT\_TX\_EMPTY.

**Parameters**

<i>instance</i>	LPI2C peripheral instance number
<i>txBuff</i>	pointer to the data to be transferred
<i>txSize</i>	length in bytes of the data to be transferred

**Returns**

Error or success status returned by API

Definition at line 2192 of file lpi2c\_driver.c.

## 16.80 LPIT Driver

### 16.80.1 Detailed Description

Low Power Interrupt Timer Peripheral Driver.

#### Hardware background

Each LPIT timer channel can be configured to run in one of 4 modes:

**32-bit Periodic Counter:** In this mode the counter will load and then decrement down to zero. It will then set the timer interrupt flag and assert the output pre-trigger.

**Dual 16-bit Periodic Counter:** In this mode, the counter will load and then the lower 16-bits will decrement down to zero, which will assert the output pre-trigger. The upper 16-bits will then decrement down to zero, which will negate the output pre-trigger and set the timer interrupt flag.

**32-bit Trigger Accumulator:** In this mode, the counter will load on the first trigger rising edge and then decrement down to zero on each trigger rising edge. It will then set the timer interrupt flag and assert the output pre-trigger.

**32-bit Trigger Input Capture:** In this mode, the counter will load with 0xFFFF\_FFFF and then decrement down to zero. If a trigger rising edge is detected, it will store the inverse of the current counter value in the load value register, set the timer interrupt flag and assert the output pre-trigger.

In these modes, the timer channel operation is further controlled by Trigger Control bits (TSOT, TSOI, TROT) which control the load, reload, start and restart of the timer channels.

#### Driver consideration

The Driver uses structures for configuration. Each structure contains members that are specific to its respective functionality. There are [lpit\\_user\\_config\\_t](#) and [lpit\\_user\\_channel\\_config\\_t](#).

#### Interrupt handling

Each LPIT timer channel has a corresponding interrupt handler. The LPIT Driver does not define interrupt handler internally. These interrupt handler methods can be defined by the user application. There are two ways to add an LPIT interrupt handler:

1. Using the weak symbols defined by start-up code. If the methods `LPITx_Handler(void)` (x denotes instance number) are not defined, the linker use a default ISR. An error will be generated if methods with the same name are defined multiple times. This method works regardless of the placement of the interrupt vector table (Flash or RAM).
2. Using the Interrupt Manager's `INT_SYS_InstallHandler()` method. This can be used to dynamically change the ISR at run-time. This method works only if the interrupt vector table is located in RAM.

#### Clocking configuration

The LPIT Driver does not handle clock setup (from PCC) configuration. This is handled by the Clock Manager. The driver assumes that clock configurations have been made, so it is the user's responsibility to set up clocking and pin configurations correctly.

#### Basic operations

1. Pre-Initialization information of LPIT module
  - Before using the LPIT driver, the protocol clock of the module must be configured by the application using PCC module.
  - Configures Trigger MUX Control (TRGMUX) if want to use external trigger for LPIT module.
  - Configures different peripherals if want to use them in LPIT interrupt routine.
  - Provides configuration data structure to LPIT initialization API.
2. To initialize the LPIT module, just call the [LPIT\\_DRV\\_Init\(\)](#) function with the user configuration data structure. This function configures LPIT module operation when MCU enters DEBUG and DOZE (Low power mode) modes and enables LPIT module. This function must be called firstly.

In the following code, LPIT module is initialized to continue to run when MCU enters both Debug and DOZE modes.

```
#define BOARD_LPIT_INSTANCE OU
/* LPIT module configuration structure */
lpit_user_config_t lpitconfig =
{
    .enableRunInDebug = true,
    .enableRunInDoze = true
};
/* Initializes the LPIT module. */
LPIT_DRV_Init(BOARD_LPIT_INSTANCE, &lpitconfig);
```

3. After calling the `LPIT_DRV_Init()` function, call `LPIT_DRV_InitChannel()` function with user channel configuration structure to initialize timer channel.

This function configures timer channel chaining, timer channel mode, timer channel period, interrupt generation, trigger source, trigger select, reload on trigger, stop on interrupt and start on trigger. In the following code, timer channel is initialized with the channel chaining is disabled, interrupt generation is enabled, operation mode is 32 bit periodic counter mode, trigger source is external, reload on trigger is disabled, stop on interrupt is disabled, start on trigger is disabled and timer period is 1 second. Note that:

- Trigger select is not effective if trigger source is external.
- Timer channel period must be suitable for operation mode.
- The timer channel 0 can not be chained.

```
/* Channel 0 configuration structure */
lpit_user_channel_config_t chnlconfig =
{
    .timerMode = LPIT_PERIODIC_COUNTER,
    .periodUnits = LPIT_PERIOD_UNITS_MICROSECONDS,
    .period = 1000000U,
    .triggerSource = LPIT_TRIGGER_SOURCE_INTERNAL,
    .triggerSelect = 1U,
    .enableReloadOnTrigger = false,
    .enableStopOnInterrupt = false,
    .enableStartOnTrigger = false,
    .chainChannel = false,
    .isInterruptEnabled = true
};
/* Initializes the channel 0 */
LPIT_DRV_InitChannel(BOARD_LPIT_INSTANCE, 0, &chnlconfig);
```

4. To reconfigure timer channel period, just call `LPIT_DRV_SetTimerPeriodByUs()` or `LPIT_DRV_SetTimerPeriodByCount()` with corresponding new period. In the following code, the timer channel period is reconfigured with new period in count unit.

```
/* Reconfigures timer channel period with new period of 10000 count*/
LPIT_DRV_SetTimerPeriodByCount(BOARD_LPIT_INSTANCE, 0, 10000);
```

5. To start timer channel counting, just call `LPIT_DRV_StartTimerChannels()` with timer channels starting mask. In the following code, the timer channel 0 is started with the mask of 0x1U.

```
/* Starts channel 0 counting*/
LPIT_DRV_StartTimerChannels(BOARD_LPIT_INSTANCE, 0x1U);
```

6. To stop timer channel counting, just call `LPIT_DRV_StopTimerChannels()` with timer channels stopping mask. In the following code, the timer channel 0 is stopped with the mask of 0x1U.

```
/* Stops channel 0 counting*/
LPIT_DRV_StopTimerChannels(BOARD_LPIT_INSTANCE, 0x1U);
```

7. To disable LPIT module, just call `LPIT_DRV_Deinit()`.

```
/* Disables LPIT module*/
LPIT_DRV_Deinit(BOARD_LPIT_INSTANCE);
```

## API

Some of the features exposed by the API are targeted specifically for timer channel mode. For example, set/get timer period in dual 16 mode function makes sense if timer channel mode is dual 16 mode, so therefor it is restricted for use in other modes.

For any invalid configuration the functions will either return an error code or trigger DEV\_ASSERT (if enabled). For more details, please refer to each function description.

### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\drivers\src\lpit_driver.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc\
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

[Clock Manager Interrupt Manager \(Interrupt\)](#)

## Data Structures

- struct [lpit\\_user\\_config\\_t](#)  
*LPIT configuration structure. [More...](#)*
- struct [lpit\\_user\\_channel\\_config\\_t](#)  
*Structure to configure the channel timer. [More...](#)*

## Macros

- #define [MAX\\_PERIOD\\_COUNT](#) (0xFFFFFFFFU)  
*Max period in count of all operation mode except for dual 16 bit periodic counter mode.*
- #define [MAX\\_PERIOD\\_COUNT\\_IN\\_DUAL\\_16BIT\\_MODE](#) (0x1FFFEU)  
*Max period in count of dual 16 bit periodic counter mode.*
- #define [MAX\\_PERIOD\\_COUNT\\_16\\_BIT](#) (0xFFFFU)  
*Max count of 16 bit.*

## Enumerations

- enum [lpit\\_timer\\_modes\\_t](#) { [LPIT\\_PERIODIC\\_COUNTER](#) = 0x00U, [LPIT\\_DUAL\\_PERIODIC\\_COUNTER](#) = 0x01U, [LPIT\\_TRIGGER\\_ACCUMULATOR](#) = 0x02U, [LPIT\\_INPUT\\_CAPTURE](#) = 0x03U }  
*Mode options available for the LPIT timer Implements : [lpit\\_timer\\_modes\\_t](#) Class.*
- enum [lpit\\_trigger\\_source\\_t](#) { [LPIT\\_TRIGGER\\_SOURCE\\_EXTERNAL](#) = 0x00U, [LPIT\\_TRIGGER\\_SOURCE\\_INTERNAL](#) = 0x01U }  
*Trigger source options.*
- enum [lpit\\_period\\_units\\_t](#) { [LPIT\\_PERIOD\\_UNITS\\_COUNTS](#) = 0x00U, [LPIT\\_PERIOD\\_UNITS\\_MICROSECONDS](#) = 0x01U }  
*Unit options for LPIT period.*

### Initialization and De-initialization

- void [LPIT\\_DRV\\_GetDefaultConfig](#) ([lpit\\_user\\_config\\_t](#) \*const config)  
*Gets the default LPIT configuration.*
- void [LPIT\\_DRV\\_GetDefaultChanConfig](#) ([lpit\\_user\\_channel\\_config\\_t](#) \*const config)  
*Gets the default timer channel configuration.*
- void [LPIT\\_DRV\\_Init](#) ([uint32\\_t](#) instance, const [lpit\\_user\\_config\\_t](#) \*userConfig)  
*Initializes the LPIT module.*
- void [LPIT\\_DRV\\_Deinit](#) ([uint32\\_t](#) instance)  
*De-Initializes the LPIT module.*
- status\_t [LPIT\\_DRV\\_InitChannel](#) ([uint32\\_t](#) instance, [uint32\\_t](#) channel, const [lpit\\_user\\_channel\\_config\\_t](#) \*userChannelConfig)  
*Initializes the LPIT channel.*

### Timer Start and Stop

- void [LPIT\\_DRV\\_StartTimerChannels](#) ([uint32\\_t](#) instance, [uint32\\_t](#) mask)  
*Starts the timer channel counting.*
- void [LPIT\\_DRV\\_StopTimerChannels](#) ([uint32\\_t](#) instance, [uint32\\_t](#) mask)  
*Stops the timer channel counting.*

### Timer Period

- status\_t [LPIT\\_DRV\\_SetTimerPeriodByUs](#) ([uint32\\_t](#) instance, [uint32\\_t](#) channel, [uint32\\_t](#) periodUs)  
*Sets the timer channel period in microseconds.*
- status\_t [LPIT\\_DRV\\_SetTimerPeriodInDual16ModeByUs](#) ([uint32\\_t](#) instance, [uint32\\_t](#) channel, [uint16\\_t](#) periodHigh, [uint16\\_t](#) periodLow)  
*Sets the timer channel period in microseconds.*
- [uint64\\_t](#) [LPIT\\_DRV\\_GetTimerPeriodByUs](#) ([uint32\\_t](#) instance, [uint32\\_t](#) channel)  
*Gets the timer channel period in microseconds.*
- [uint64\\_t](#) [LPIT\\_DRV\\_GetCurrentTimerUs](#) ([uint32\\_t](#) instance, [uint32\\_t](#) channel)  
*Gets the current timer channel counting value in microseconds.*
- void [LPIT\\_DRV\\_SetTimerPeriodByCount](#) ([uint32\\_t](#) instance, [uint32\\_t](#) channel, [uint32\\_t](#) count)  
*Sets the timer channel period in count unit.*
- void [LPIT\\_DRV\\_SetTimerPeriodInDual16ModeByCount](#) ([uint32\\_t](#) instance, [uint32\\_t](#) channel, [uint16\\_t](#) periodHigh, [uint16\\_t](#) periodLow)  
*Sets the timer channel period in count unit.*
- [uint32\\_t](#) [LPIT\\_DRV\\_GetTimerPeriodByCount](#) ([uint32\\_t](#) instance, [uint32\\_t](#) channel)  
*Gets the current timer channel period in count unit.*
- [uint32\\_t](#) [LPIT\\_DRV\\_GetCurrentTimerCount](#) ([uint32\\_t](#) instance, [uint32\\_t](#) channel)  
*Gets the current timer channel counting value in count.*

### Interrupt

- void [LPIT\\_DRV\\_EnableTimerChannelInterrupt](#) ([uint32\\_t](#) instance, [uint32\\_t](#) mask)  
*Enables the interrupt generation of timer channel.*
- void [LPIT\\_DRV\\_DisableTimerChannelInterrupt](#) ([uint32\\_t](#) instance, [uint32\\_t](#) mask)  
*Disables the interrupt generation of timer channel.*
- [uint32\\_t](#) [LPIT\\_DRV\\_GetInterruptFlagTimerChannels](#) ([uint32\\_t](#) instance, [uint32\\_t](#) mask)  
*Gets the current interrupt flag of timer channels.*
- void [LPIT\\_DRV\\_ClearInterruptFlagTimerChannels](#) ([uint32\\_t](#) instance, [uint32\\_t](#) mask)  
*Clears the interrupt flag of timer channels.*

## 16.80.2 Data Structure Documentation

### 16.80.2.1 struct `lpit_user_config_t`

LPIT configuration structure.

This structure holds the configuration settings for the LPIT peripheral to enable or disable LPIT module in DEBUG and DOZE mode Implements : `lpit_user_config_t_Class`

Definition at line 108 of file `lpit_driver.h`.

#### Data Fields

- bool [enableRunInDebug](#)
- bool [enableRunInDoze](#)

#### Field Documentation

##### 16.80.2.1.1 bool `enableRunInDebug`

True: Timer channels continue to run in debug mode False: Timer channels stop in debug mode

Definition at line 110 of file `lpit_driver.h`.

##### 16.80.2.1.2 bool `enableRunInDoze`

True: Timer channels continue to run in doze mode False: Timer channels stop in doze mode

Definition at line 112 of file `lpit_driver.h`.

### 16.80.2.2 struct `lpit_user_channel_config_t`

Structure to configure the channel timer.

This structure holds the configuration settings for the LPIT timer channel Implements : `lpit_user_channel_config_t_Class`

Definition at line 121 of file `lpit_driver.h`.

#### Data Fields

- [lpit\\_timer\\_modes\\_t](#) `timerMode`
- [lpit\\_period\\_units\\_t](#) `periodUnits`
- [uint32\\_t](#) `period`
- [lpit\\_trigger\\_source\\_t](#) `triggerSource`
- [uint32\\_t](#) `triggerSelect`
- bool [enableReloadOnTrigger](#)
- bool [enableStopOnInterrupt](#)
- bool [enableStartOnTrigger](#)
- bool [chainChannel](#)
- bool [isInterruptEnabled](#)

#### Field Documentation

##### 16.80.2.2.1 bool `chainChannel`

Channel chaining enable

Definition at line 137 of file `lpit_driver.h`.

##### 16.80.2.2.2 bool `enableReloadOnTrigger`

True: Timer channel will reload on selected trigger False: Timer channel will not reload on selected trigger

Definition at line 129 of file `lpit_driver.h`.



**16.80.2.2.3 bool enableStartOnTrigger**

True: Timer channel starts to decrement when rising edge on selected trigger is detected. False: Timer starts to decrement immediately based on restart condition

Definition at line 133 of file lpit\_driver.h.

**16.80.2.2.4 bool enableStopOnInterrupt**

True: Timer will stop after timeout False: Timer channel does not stop after timeout

Definition at line 131 of file lpit\_driver.h.

**16.80.2.2.5 bool isInterruptEnabled**

Timer channel interrupt generation enable

Definition at line 138 of file lpit\_driver.h.

**16.80.2.2.6 uint32\_t period**

Period of timer channel

Definition at line 125 of file lpit\_driver.h.

**16.80.2.2.7 lpit\_period\_units\_t periodUnits**

Timer period value units

Definition at line 124 of file lpit\_driver.h.

**16.80.2.2.8 lpit\_timer\_modes\_t timerMode**

Operation mode of timer channel

Definition at line 123 of file lpit\_driver.h.

**16.80.2.2.9 uint32\_t triggerSelect**

Selects one trigger from the internal trigger sources this field makes sense if trigger source is internal

Definition at line 127 of file lpit\_driver.h.

**16.80.2.2.10 lpit\_trigger\_source\_t triggerSource**

Selects between internal and external trigger sources

Definition at line 126 of file lpit\_driver.h.

**16.80.3 Macro Definition Documentation****16.80.3.1 #define MAX\_PERIOD\_COUNT (0xFFFFFFFFU)**

Max period in count of all operation mode except for dual 16 bit periodic counter mode.

Definition at line 58 of file lpit\_driver.h.

**16.80.3.2 #define MAX\_PERIOD\_COUNT\_16\_BIT (0xFFFFU)**

Max count of 16 bit.

Definition at line 62 of file lpit\_driver.h.

### 16.80.3.3 #define MAX\_PERIOD\_COUNT\_IN\_DUAL\_16BIT\_MODE (0x1FFFEU)

Max period in count of dual 16 bit periodic counter mode.

Definition at line 60 of file lpit\_driver.h.

## 16.80.4 Enumeration Type Documentation

### 16.80.4.1 enum lpit\_period\_units\_t

Unit options for LPIT period.

This is used to determine unit of timer period Implements : lpit\_period\_units\_t\_Class

Enumerator

**LPIT\_PERIOD\_UNITS\_COUNTS** Period value unit is count

**LPIT\_PERIOD\_UNITS\_MICROSECONDS** Period value unit is microsecond

Definition at line 95 of file lpit\_driver.h.

### 16.80.4.2 enum lpit\_timer\_modes\_t

Mode options available for the LPIT timer Implements : lpit\_timer\_modes\_t\_Class.

Enumerator

**LPIT\_PERIODIC\_COUNTER** 32-bit Periodic Counter

**LPIT\_DUAL\_PERIODIC\_COUNTER** Dual 16-bit Periodic Counter

**LPIT\_TRIGGER\_ACCUMULATOR** 32-bit Trigger Accumulator

**LPIT\_INPUT\_CAPTURE** 32-bit Trigger Input Capture

Definition at line 68 of file lpit\_driver.h.

### 16.80.4.3 enum lpit\_trigger\_source\_t

Trigger source options.

This is used for both internal and external trigger sources. The actual trigger options available is SoC specific, user should refer to the reference manual. Implements : lpit\_trigger\_source\_t\_Class

Enumerator

**LPIT\_TRIGGER\_SOURCE\_EXTERNAL** Use external trigger

**LPIT\_TRIGGER\_SOURCE\_INTERNAL** Use internal trigger

Definition at line 83 of file lpit\_driver.h.

## 16.80.5 Function Documentation

### 16.80.5.1 void LPIT\_DRV\_ClearInterruptFlagTimerChannels ( uint32\_t instance, uint32\_t mask )

Clears the interrupt flag of timer channels.

This function clears the interrupt flag of timer channels after their interrupt event occurred.

## Parameters

in	<i>instance</i>	LPIT module instance number
in	<i>mask</i>	<p>The interrupt flag clearing mask that decides which channels will be cleared interrupt flag</p> <ul style="list-style-type: none"> <li>• For example: <ul style="list-style-type: none"> <li>– with mask = 0x01u then the interrupt flag of channel 0 only will be cleared</li> <li>– with mask = 0x02u then the interrupt flag of channel 1 only will be cleared</li> <li>– with mask = 0x03u then the interrupt flags of channel 0 and channel 1 will be cleared</li> </ul> </li> </ul>

Definition at line 744 of file lpit\_driver.c.

#### 16.80.5.2 void LPIT\_DRV\_Deinit ( uint32\_t *instance* )

De-Initializes the LPIT module.

This function disables LPIT module. In order to use the LPIT module again, LPIT\_DRV\_Init must be called.

## Parameters

in	<i>instance</i>	LPIT module instance number
----	-----------------	-----------------------------

Definition at line 177 of file lpit\_driver.c.

#### 16.80.5.3 void LPIT\_DRV\_DisableTimerChannelInterrupt ( uint32\_t *instance*, uint32\_t *mask* )

Disables the interrupt generation of timer channel.

This function allows disabling interrupt generation of timer channel when timeout occurs or input trigger occurs.

## Parameters

in	<i>instance</i>	LPIT module instance number
in	<i>mask</i>	<p>The mask that decides which channels will be disable interrupt.</p> <ul style="list-style-type: none"> <li>• For example: <ul style="list-style-type: none"> <li>– with mask = 0x01u then the interrupt of channel 0 will be disable</li> <li>– with mask = 0x02u then the interrupt of channel 1 will be disable</li> <li>– with mask = 0x03u then the interrupt of channel 0 and channel 1 will be disable</li> </ul> </li> </ul>

Definition at line 701 of file lpit\_driver.c.

#### 16.80.5.4 void LPIT\_DRV\_EnableTimerChannelInterrupt ( uint32\_t *instance*, uint32\_t *mask* )

Enables the interrupt generation of timer channel.

This function allows enabling interrupt generation of timer channel when timeout occurs or input trigger occurs.

## Parameters

in	<i>instance</i>	LPIT module instance number.
----	-----------------	------------------------------

<i>in</i>	<i>mask</i>	<p>The mask that decides which channels will be enabled interrupt.</p> <ul style="list-style-type: none"> <li>• For example: <ul style="list-style-type: none"> <li>– with mask = 0x01u then the interrupt of channel 0 will be enabled</li> <li>– with mask = 0x02u then the interrupt of channel 1 will be enabled</li> <li>– with mask = 0x03u then the interrupt of channel 0 and channel 1 will be enabled</li> </ul> </li> </ul>
-----------	-------------	--

Definition at line 680 of file lpit\_driver.c.

#### 16.80.5.5 uint32\_t LPIT\_DRV\_GetCurrentTimerCount ( uint32\_t *instance*, uint32\_t *channel* )

Gets the current timer channel counting value in count.

This function returns the real-time timer channel counting value, the value in a range from 0 to timer channel period. Need to make sure the running time does not exceed the timer channel period.

##### Parameters

<i>in</i>	<i>instance</i>	LPIT module instance number
<i>in</i>	<i>channel</i>	Timer channel number

##### Returns

Current timer channel counting value in count

Definition at line 648 of file lpit\_driver.c.

#### 16.80.5.6 uint64\_t LPIT\_DRV\_GetCurrentTimerUs ( uint32\_t *instance*, uint32\_t *channel* )

Gets the current timer channel counting value in microseconds.

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. The return counting value here makes sense if the operation mode of timer channel is 32 bit periodic counter or dual 16 bit periodic counter or 32-bit trigger input capture. Need to make sure the running time will not exceed the timer channel period.

##### Parameters

<i>in</i>	<i>instance</i>	LPIT module instance number
<i>in</i>	<i>channel</i>	Timer channel number

##### Returns

Current timer channel counting value in microseconds

Definition at line 512 of file lpit\_driver.c.

#### 16.80.5.7 void LPIT\_DRV\_GetDefaultChanConfig ( lpit\_user\_channel\_config\_t \*const *config* )

Gets the default timer channel configuration.

This function gets the default timer channel configuration structure, with the following settings:

- Timer mode: 32-bit Periodic Counter
- Period unit: Period value unit is microsecond
- Period: 1000000 microseconds(1 second)
- Trigger sources: External trigger

- Trigger select: Trigger from channel 0
- Reload on trigger: Disable
- Stop on interrupt : Disable
- Start on trigger: Disable
- Channel chaining: Disable
- Interrupt generating: Enable

**Parameters**

out	config	The channel configuration structure
-----	--------	-------------------------------------

Definition at line 102 of file lpit\_driver.c.

#### 16.80.5.8 void LPIT\_DRV\_GetDefaultConfig ( lpit\_user\_config\_t \*const config )

Gets the default LPIT configuration.

This function gets default LPIT module configuration structure, with the following settings:

- PIT runs in debug mode: Disable
- PIT runs in doze mode: Disable

**Parameters**

out	config	The configuration structure
-----	--------	-----------------------------

Definition at line 87 of file lpit\_driver.c.

#### 16.80.5.9 uint32\_t LPIT\_DRV\_GetInterruptFlagTimerChannels ( uint32\_t instance, uint32\_t mask )

Gets the current interrupt flag of timer channels.

This function gets the current interrupt flag of timer channels. In compare modes, the flag sets to 1 at the end of the timer period. In capture modes, the flag sets to 1 when the trigger asserts.

**Parameters**

in	instance	LPIT module instance number.
in	mask	<p>The interrupt flag getting mask that decides which channels will be got interrupt flag.</p> <ul style="list-style-type: none"> <li>• For example: <ul style="list-style-type: none"> <li>– with mask = 0x01u then the interrupt flag of channel 0 only will be got</li> <li>– with mask = 0x02u then the interrupt flag of channel 1 only will be got</li> <li>– with mask = 0x03u then the interrupt flags of channel 0 and channel 1 will be got</li> </ul> </li> </ul>

**Returns**

Current the interrupt flag of timer channels

Definition at line 723 of file lpit\_driver.c.

16.80.5.10 `uint32_t LPIT_DRV_GetTimerPeriodByCount ( uint32_t instance, uint32_t channel )`

Gets the current timer channel period in count unit.

This function returns current period of timer channel given as argument.

**Parameters**

in	<i>instance</i>	LPIT module instance number
in	<i>channel</i>	Timer channel number

**Returns**

Timer channel period in count unit

Definition at line 614 of file lpit\_driver.c.

#### 16.80.5.11 uint64\_t LPIT\_DRV\_GetTimerPeriodByUs ( uint32\_t *instance*, uint32\_t *channel* )

Gets the timer channel period in microseconds.

This function gets the timer channel period in microseconds. The returned period here makes sense if the operation mode of timer channel is 32 bit periodic counter or dual 16 bit periodic counter.

**Parameters**

in	<i>instance</i>	LPIT module instance number
in	<i>channel</i>	Timer channel number

**Returns**

Timer channel period in microseconds

Definition at line 453 of file lpit\_driver.c.

#### 16.80.5.12 void LPIT\_DRV\_Init ( uint32\_t *instance*, const lpit\_user\_config\_t \* *userConfig* )

Initializes the LPIT module.

This function resets LPIT module, enables the LPIT module, configures LPIT module operation in Debug and DOZE mode. The LPIT configuration structure shall be passed as arguments. This configuration structure affects all timer channels. This function should be called before calling any other LPIT driver function.

This is an example demonstrating how to define a LPIT configuration structure:

```
1 lpit_user_config_t lpitInit =
2 {
3     .enableRunInDebug = false,
4     .enableRunInDoze = true
5 };
```

**Parameters**

in	<i>instance</i>	LPIT module instance number.
in	<i>userConfig</i>	Pointer to LPIT configuration structure.

Definition at line 130 of file lpit\_driver.c.

#### 16.80.5.13 status\_t LPIT\_DRV\_InitChannel ( uint32\_t *instance*, uint32\_t *channel*, const lpit\_user\_channel\_config\_t \* *userChannelConfig* )

Initializes the LPIT channel.

This function initializes the LPIT timers by using a channel, this function configures timer channel chaining, timer channel mode, timer channel period, interrupt generation, trigger source, trigger select, reload on trigger, stop on interrupt and start on trigger. The timer channel number and its configuration structure shall be passed as arguments. Timer channels do not start counting by default after calling this function. The function LPIT\_DRV\_StartTimerChannels must be called to start the timer channel counting. In order to re-configures the period, call the LPIT\_DRV\_SetTimerPeriodByUs or LPIT\_DRV\_SetTimerPeriodByCount.

This is an example demonstrating how to define a LPIT channel configuration structure:

```

1 lpit_user_channel_config_t lpitTestInit =
2 {
3   .timerMode = LPIT_PERIODIC_COUNTER,
4   .periodUnits = LPTT_PERIOD_UNITS_MICROSECONDS,
5   .period = 1000000U,
6   .triggerSource = LPIT_TRIGGER_SOURCE_INTERNAL,
7   .triggerSelect = 1U,
8   .enableReloadOnTrigger = false,
9   .enableStopOnInterrupt = false,
10  .enableStartOnTrigger = false,
11  .chainChannel = false,
12  .isInterruptEnabled = true
13 };

```

#### Parameters

in	<i>instance</i>	LPIT module instance number
in	<i>channel</i>	Timer channel number
in	<i>userChannelConfig</i>	Pointer to LPIT channel configuration structure

#### Returns

##### Operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: The channel 0 is chained.
- STATUS\_ERROR: The input period is invalid.

Definition at line 204 of file lpit\_driver.c.

**16.80.5.14** void LPIT\_DRV\_SetTimerPeriodByCount ( uint32\_t *instance*, uint32\_t *channel*, uint32\_t *count* )

Sets the timer channel period in count unit.

This function sets the timer channel period in count unit. The counter period of a running timer channel can be modified by first setting a new load value, the value will be loaded after the timer channel expires. To abort the current cycle and start a timer channel period with the new value, the timer channel must be disabled and enabled again.

#### Parameters

in	<i>instance</i>	LPIT module instance number
in	<i>channel</i>	Timer channel number
in	<i>count</i>	Timer channel period in count unit

Definition at line 560 of file lpit\_driver.c.

**16.80.5.15** status\_t LPIT\_DRV\_SetTimerPeriodByUs ( uint32\_t *instance*, uint32\_t *channel*, uint32\_t *periodUs* )

Sets the timer channel period in microseconds.

This function sets the timer channel period in microseconds when timer channel mode is 32 bit periodic or dual 16 bit counter mode. The period range depends on the frequency of the LPIT functional clock and operation mode of timer channel. If the required period is out of range, use the suitable mode if applicable. This function is only valid for one single channel.

#### Parameters

in	<i>instance</i>	LPIT module instance number
in	<i>channel</i>	Timer channel number



<i>in</i>	<i>periodUs</i>	Timer channel period in microseconds
-----------	-----------------	--------------------------------------

**Returns**

## Operation status

- STATUS\_SUCCESS: Input period of timer channel is valid.
- STATUS\_ERROR: Input period of timer channel is invalid.

Definition at line 329 of file lpit\_driver.c.

**16.80.5.16** void LPIT\_DRV\_SetTimerPeriodInDual16ModeByCount ( uint32\_t *instance*, uint32\_t *channel*, uint16\_t *periodHigh*, uint16\_t *periodLow* )

Sets the timer channel period in count unit.

This function sets the timer channel period in count unit when timer channel mode is dual 16 periodic counter mode. The counter period of a running timer channel can be modified by first setting a new load value, the value will be loaded after the timer channel expires. To abort the current cycle and start a timer channel period with the new value, the timer channel must be disabled and enabled again.

**Parameters**

<i>in</i>	<i>instance</i>	LPIT module instance number
<i>in</i>	<i>channel</i>	Timer channel number
<i>in</i>	<i>periodHigh</i>	Period of higher 16 bit in count unit
<i>in</i>	<i>periodLow</i>	Period of lower 16 bit in count unit

Definition at line 588 of file lpit\_driver.c.

**16.80.5.17** status\_t LPIT\_DRV\_SetTimerPeriodInDual16ModeByUs ( uint32\_t *instance*, uint32\_t *channel*, uint16\_t *periodHigh*, uint16\_t *periodLow* )

Sets the timer channel period in microseconds.

This function sets the timer channel period in microseconds when timer channel mode is dual 16 bit periodic counter mode. The period range depends on the frequency of the LPIT functional clock and operation mode of timer channel. If the required period is out of range, use the suitable mode if applicable. This function is only valid for one single channel.

**Parameters**

<i>in</i>	<i>instance</i>	LPIT module instance number
<i>in</i>	<i>channel</i>	Timer channel number
<i>in</i>	<i>periodHigh</i>	Period of higher 16 bit in microseconds
<i>in</i>	<i>periodLow</i>	Period of lower 16 bit in microseconds

**Returns**

## Operation status

- STATUS\_SUCCESS: Input period of timer channel is valid.
- STATUS\_ERROR: Input period of timer channel is invalid.

Definition at line 400 of file lpit\_driver.c.

**16.80.5.18** void LPIT\_DRV\_StartTimerChannels ( uint32\_t *instance*, uint32\_t *mask* )

Starts the timer channel counting.

This function allows starting timer channels simultaneously . After calling this function, timer channels are going operate depend on mode and control bits which controls timer channel start, reload and restart.

**Parameters**

<i>in</i>	<i>instance</i>	LPIT module instance number
<i>in</i>	<i>mask</i>	Timer channels starting mask that decides which channels will be started <ul style="list-style-type: none"> <li>• For example:               <ul style="list-style-type: none"> <li>– with mask = 0x01U then channel 0 will be started</li> <li>– with mask = 0x02U then channel 1 will be started</li> <li>– with mask = 0x03U then channel 0 and channel 1 will be started</li> </ul> </li> </ul>

Definition at line 279 of file lpit\_driver.c.

**16.80.5.19** void LPIT\_DRV\_StopTimerChannels ( uint32\_t *instance*, uint32\_t *mask* )

Stops the timer channel counting.

This function allows stop timer channels simultaneously from counting. Timer channels reload their periods respectively after the next time they call the LPIT\_DRV\_StartTimerChannels. Note that: In 32-bit Trigger Accumulator mode, the counter will load on the first trigger rising edge.

**Parameters**

<i>in</i>	<i>instance</i>	LPIT module instance number
<i>in</i>	<i>mask</i>	Timer channels stopping mask that decides which channels will be stopped <ul style="list-style-type: none"> <li>• For example:               <ul style="list-style-type: none"> <li>– with mask = 0x01U then channel 0 will be stopped</li> <li>– with mask = 0x02U then channel 1 will be stopped</li> <li>– with mask = 0x03U then channel 0 and channel 1 will be stopped</li> </ul> </li> </ul>

Definition at line 303 of file lpit\_driver.c.

## 16.81 LPSPI Driver

### 16.81.1 Detailed Description

Low Power Serial Peripheral Interface Peripheral Driver.

#### Data Structures

- struct [lpspi\\_master\\_config\\_t](#)  
Data structure containing information about a device on the SPI bus. [More...](#)
- struct [lpspi\\_state\\_t](#)  
Runtime state structure for the LPSPI master driver. [More...](#)
- struct [lpspi\\_slave\\_config\\_t](#)  
User configuration structure for the SPI slave driver. Implements : [lpspi\\_slave\\_config\\_t\\_Class](#). [More...](#)

#### Enumerations

- enum [lpspi\\_which\\_pcs\\_t](#) { [LPSPI\\_PCS0](#) = 0U, [LPSPI\\_PCS1](#) = 1U, [LPSPI\\_PCS2](#) = 2U, [LPSPI\\_PCS3](#) = 3U }  
LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure). Implements : [lpspi\\_which\\_pcs\\_t\\_Class](#).
- enum [lpspi\\_signal\\_polarity\\_t](#) { [LPSPI\\_ACTIVE\\_HIGH](#) = 1U, [LPSPI\\_ACTIVE\\_LOW](#) = 0U }  
LPSPI Signal (PCS and Host Request) Polarity configuration. Implements : [lpspi\\_signal\\_polarity\\_t\\_Class](#).
- enum [lpspi\\_clock\\_phase\\_t](#) { [LPSPI\\_CLOCK\\_PHASE\\_1ST\\_EDGE](#) = 0U, [LPSPI\\_CLOCK\\_PHASE\\_2ND\\_EDGE](#) = 1U }  
LPSPI clock phase configuration. Implements : [lpspi\\_clock\\_phase\\_t\\_Class](#).
- enum [lpspi\\_sck\\_polarity\\_t](#) { [LPSPI\\_SCK\\_ACTIVE\\_HIGH](#) = 0U, [LPSPI\\_SCK\\_ACTIVE\\_LOW](#) = 1U }  
LPSPI Clock Signal (SCK) Polarity configuration. Implements : [lpspi\\_sck\\_polarity\\_t\\_Class](#).
- enum [lpspi\\_transfer\\_type](#) { [LPSPI\\_USING\\_DMA](#) = 0, [LPSPI\\_USING\\_INTERRUPTS](#) }  
Type of LPSPI transfer (based on interrupts or DMA). Implements : [lpspi\\_transfer\\_type\\_Class](#).
- enum [transfer\\_status\\_t](#) { [LPSPI\\_TRANSFER\\_OK](#) = 0U, [LPSPI\\_TRANSMIT\\_FAIL](#), [LPSPI\\_RECEIVE\\_FAIL](#) }  
Type of error reported by LPSPI.

#### Functions

- void [LPSPI\\_DRV\\_SlaveIRQHandler](#) (uint32\_t instance)  
Interrupt handler for LPSPI slave mode. This handler uses the buffers stored in the [lpspi\\_master\\_state\\_t](#) structs to transfer data.
- void [LPSPI\\_DRV\\_IRQHandler](#) (uint32\_t instance)  
The function [LPSPI\\_DRV\\_IRQHandler](#) passes IRQ control to either the master or slave driver.
- void [LPSPI\\_DRV\\_FillupTxBuffer](#) (uint32\_t instance)  
The function [LPSPI\\_DRV\\_FillupTxBuffer](#) writes data in TX hardware buffer depending on driver state and number of bytes remained to send.
- void [LPSPI\\_DRV\\_ReadRXBuffer](#) (uint32\_t instance)  
The function [LPSPI\\_DRV\\_ReadRXBuffer](#) reads data from RX hardware buffer and writes this data in RX software buffer.
- void [LPSPI\\_DRV\\_DisableTEIEInterrupts](#) (uint32\_t instance)  
Disable the TEIE interrupts at the end of a transfer. Disable the interrupts and clear the status for transmit/receive errors.
- void [LPSPI\\_DRV\\_SlaveGetDefaultConfig](#) ([lpspi\\_slave\\_config\\_t](#) \*spiConfig)  
Return default configuration for SPI master.
- status\_t [LPSPI\\_DRV\\_SlaveInit](#) (uint32\_t instance, [lpspi\\_state\\_t](#) \*lpspiState, const [lpspi\\_slave\\_config\\_t](#) \*slaveConfig)  
Initializes a LPSPI instance for a slave mode operation, using interrupt mechanism.

- status\_t [LPSPi\\_DRV\\_SlaveDeinit](#) (uint32\_t instance)  
*Shuts down an LPSPi instance interrupt mechanism.*
- status\_t [LPSPi\\_DRV\\_SlaveTransferBlocking](#) (uint32\_t instance, const uint8\_t \*sendBuffer, uint8\_t \*receiveBuffer, uint16\_t transferByteCount, uint32\_t timeout)  
*Transfers data on LPSPi bus using a blocking call.*
- status\_t [LPSPi\\_DRV\\_SlaveTransfer](#) (uint32\_t instance, const uint8\_t \*sendBuffer, uint8\_t \*receiveBuffer, uint16\_t transferByteCount)  
*Starts the transfer data on LPSPi bus using a non-blocking call.*
- status\_t [LPSPi\\_DRV\\_SlaveAbortTransfer](#) (uint32\_t instance)  
*Aborts the transfer that started by a non-blocking call transfer function.*
- status\_t [LPSPi\\_DRV\\_SlaveGetTransferStatus](#) (uint32\_t instance, uint32\_t \*bytesRemained)  
*Returns whether the previous transfer is finished.*
- void [LPSPi0\\_IRQHandler](#) (void)  
*This function is the implementation of LPSPi0 handler named in startup code.*
- void [LPSPi1\\_IRQHandler](#) (void)  
*This function is the implementation of LPSPi1 handler named in startup code.*
- void [LPSPi2\\_IRQHandler](#) (void)  
*This function is the implementation of LPSPi2 handler named in startup code.*

## Variables

- LPSPi\_Type \* [g\\_lpspiBase](#) [LPSPi\_INSTANCE\_COUNT]  
*Table of base pointers for SPI instances.*
- IRQn\_Type [g\\_lpspiIrqlId](#) [LPSPi\_INSTANCE\_COUNT]  
*Table to save LPSPi IRQ enumeration numbers defined in the CMSIS header file.*
- [lpspi\\_state\\_t](#) \* [g\\_lpspiStatePtr](#) [LPSPi\_INSTANCE\_COUNT]

## Initialization and shutdown

- void [LPSPi\\_DRV\\_MasterGetDefaultConfig](#) (lpspi\_master\_config\_t \*spiConfig)  
*Return default configuration for SPI master.*
- status\_t [LPSPi\\_DRV\\_MasterInit](#) (uint32\_t instance, [lpspi\\_state\\_t](#) \*lpspiState, const [lpspi\\_master\\_config\\_t](#) \*spiConfig)  
*Initializes a LPSPi instance for interrupt driven master mode operation.*
- status\_t [LPSPi\\_DRV\\_MasterDeinit](#) (uint32\_t instance)  
*Shuts down a LPSPi instance.*
- status\_t [LPSPi\\_DRV\\_MasterSetDelay](#) (uint32\_t instance, uint32\_t delayBetweenTransfers, uint32\_t delaySCKtoPCS, uint32\_t delayPCStoSCK)  
*Configures the LPSPi master mode bus timing delay options.*

## Bus configuration

- status\_t [LPSPi\\_DRV\\_MasterConfigureBus](#) (uint32\_t instance, const [lpspi\\_master\\_config\\_t](#) \*spiConfig, uint32\_t \*calculatedBaudRate)  
*Configures the LPSPi port physical parameters to access a device on the bus when the LPSPi instance is configured for interrupt operation.*
- status\_t [LPSPi\\_DRV\\_SetPcs](#) (uint32\_t instance, [lpspi\\_which\\_pcs\\_t](#) whichPcs, [lpspi\\_signal\\_polarity\\_t](#) polarity)  
*Select the chip to communicate with.*

**Blocking transfers**

- status\_t [LPSPi\\_DRV\\_MasterTransferBlocking](#) (uint32\_t instance, const uint8\_t \*sendBuffer, uint8\_t \*receiveBuffer, uint16\_t transferByteCount, uint32\_t timeout)

*Performs an interrupt driven blocking SPI master mode transfer.*

**Non-blocking transfers**

- status\_t [LPSPi\\_DRV\\_MasterTransfer](#) (uint32\_t instance, const uint8\_t \*sendBuffer, uint8\_t \*receiveBuffer, uint16\_t transferByteCount)

*Performs an interrupt driven non-blocking SPI master mode transfer.*

- status\_t [LPSPi\\_DRV\\_MasterGetTransferStatus](#) (uint32\_t instance, uint32\_t \*bytesRemained)

*Returns whether the previous interrupt driven transfer is completed.*

- status\_t [LPSPi\\_DRV\\_MasterAbortTransfer](#) (uint32\_t instance)

*Terminates an interrupt driven asynchronous transfer early.*

- void [LPSPi\\_DRV\\_MasterIRQHandler](#) (uint32\_t instance)

*Interrupt handler for LPSPi master mode. This handler uses the buffers stored in the `lpspi_master_state_t` structs to transfer data.*

**16.81.2 Data Structure Documentation****16.81.2.1 struct lpspi\_master\_config\_t**

Data structure containing information about a device on the SPI bus.

The user must populate these members to set up the LPSPi master and properly communicate with the SPI device.

Implements : `lpspi_master_config_t_Class`

Definition at line 49 of file `lpspi_master_driver.h`.

**Data Fields**

- uint32\_t [bitsPerSec](#)
- [lpspi\\_which\\_pcs\\_t](#) `whichPcs`
- [lpspi\\_signal\\_polarity\\_t](#) `pcsPolarity`
- bool [isPcsContinuous](#)
- uint16\_t [bitcount](#)
- uint32\_t [lpspiSrcClk](#)
- [lpspi\\_clock\\_phase\\_t](#) `clkPhase`
- [lpspi\\_sck\\_polarity\\_t](#) `clkPolarity`
- bool [lsbFirst](#)
- [lpspi\\_transfer\\_type](#) `transferType`
- uint8\_t [rxDMAChannel](#)
- uint8\_t [txDMAChannel](#)
- [spi\\_callback\\_t](#) `callback`
- void \* [callbackParam](#)

**Field Documentation****16.81.2.1.1 uint16\_t bitcount**

Number of bits/frame, minimum is 8-bits

Definition at line 55 of file `lpspi_master_driver.h`.

**16.81.2.1.2 uint32\_t bitsPerSec**

Baud rate in bits per second

Definition at line 51 of file `lpspi_master_driver.h`.

**16.81.2.1.3 spi\_callback\_t callback**

Select the callback to transfer complete

Definition at line 63 of file `lpspi_master_driver.h`.

**16.81.2.1.4 void\* callbackParam**

Select additional callback parameters if it's necessary

Definition at line 64 of file `lpspi_master_driver.h`.

**16.81.2.1.5 lpspi\_clock\_phase\_t clkPhase**

Selects which phase of clock to capture data

Definition at line 57 of file `lpspi_master_driver.h`.

**16.81.2.1.6 lpspi\_sck\_polarity\_t clkPolarity**

Selects clock polarity

Definition at line 58 of file `lpspi_master_driver.h`.

**16.81.2.1.7 bool isPcsContinuous**

Keeps PCS asserted until transfer complete

Definition at line 54 of file `lpspi_master_driver.h`.

**16.81.2.1.8 uint32\_t lpspiSrcClk**

Module source clock

Definition at line 56 of file `lpspi_master_driver.h`.

**16.81.2.1.9 bool lsbFirst**

Option to transmit LSB first

Definition at line 59 of file `lpspi_master_driver.h`.

**16.81.2.1.10 lpspi\_signal\_polarity\_t pcsPolarity**

PCS polarity

Definition at line 53 of file `lpspi_master_driver.h`.

**16.81.2.1.11 uint8\_t rxDMAChannel**

Channel number for DMA rx channel. If DMA mode isn't used this field will be ignored.

Definition at line 61 of file `lpspi_master_driver.h`.

**16.81.2.1.12 lpspi\_transfer\_type transferType**

Type of LPSPI transfer

Definition at line 60 of file `lpspi_master_driver.h`.

**16.81.2.1.13 uint8\_t txDMAChannel**

Channel number for DMA tx channel. If DMA mode isn't used this field will be ignored.

Definition at line 62 of file `lpspi_master_driver.h`.

**16.81.2.1.14 lpspi\_which\_pcs\_t whichPcs**

Selects which PCS to use

Definition at line 52 of file `lpspi_master_driver.h`.

**16.81.2.2 struct lpspi\_state\_t**

Runtime state structure for the LPSPi master driver.

This structure holds data that is used by the LPSPi 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 LPSPi master driver populates the members. Implements : `lpspi_state_t_Class`

Definition at line 124 of file `lpspi_shared_function.h`.

**Data Fields**

- `uint16_t bitsPerFrame`
- `uint16_t bytesPerFrame`
- `bool isPcsContinuous`
- `bool isBlocking`
- `uint32_t lpspiSrcClk`
- `volatile bool isTransferInProgress`
- `const uint8_t * txBuff`
- `uint8_t * rxBuff`
- `volatile uint16_t txCount`
- `volatile uint16_t rxCount`
- `volatile uint16_t txFrameCnt`
- `volatile uint16_t rxFrameCnt`
- `volatile bool lsb`
- `uint8_t fifoSize`
- `uint8_t rxDMAChannel`
- `uint8_t txDMAChannel`
- `lpspi_transfer_type transferType`
- `semaphore_t lpspiSemaphore`
- `transfer_status_t status`
- `spi_callback_t callback`
- `void * callbackParam`
- `uint32_t dummy`

**Field Documentation****16.81.2.2.1 uint16\_t bitsPerFrame**

Number of bits per frame: 8- to 4096-bits; needed for TCR programming

Definition at line 126 of file `lpspi_shared_function.h`.

**16.81.2.2.2 uint16\_t bytesPerFrame**

Number of bytes per frame: 1- to 512-bytes

Definition at line 128 of file `lpspi_shared_function.h`.

**16.81.2.2.3 spi\_callback\_t callback**

Select the callback to transfer complete

Definition at line 147 of file lpspi\_shared\_function.h.

**16.81.2.2.4 void\* callbackParam**

Select additional callback parameters if it's necessary

Definition at line 148 of file lpspi\_shared\_function.h.

**16.81.2.2.5 uint32\_t dummy**

This field is used for the cases when TX is NULL and LPSPi is in DMA mode

Definition at line 149 of file lpspi\_shared\_function.h.

**16.81.2.2.6 uint8\_t fifoSize**

RX/TX fifo size

Definition at line 141 of file lpspi\_shared\_function.h.

**16.81.2.2.7 bool isBlocking**

Save the transfer type

Definition at line 131 of file lpspi\_shared\_function.h.

**16.81.2.2.8 bool isPcsContinuous**

Option to keep chip select asserted until transfer complete; needed for TCR programming

Definition at line 129 of file lpspi\_shared\_function.h.

**16.81.2.2.9 volatile bool isTransferInProgress**

True if there is an active transfer

Definition at line 133 of file lpspi\_shared\_function.h.

**16.81.2.2.10 semaphore\_t lpspiSemaphore**

The semaphore used for blocking transfers

Definition at line 145 of file lpspi\_shared\_function.h.

**16.81.2.2.11 uint32\_t lpspiSrcClk**

Module source clock

Definition at line 132 of file lpspi\_shared\_function.h.

**16.81.2.2.12 volatile bool lsb**

True if first bit is LSB and false if first bit is MSB

Definition at line 140 of file lpspi\_shared\_function.h.

**16.81.2.2.13 uint8\_t\* rxBuff**

The buffer into which received bytes are placed

Definition at line 135 of file lpspi\_shared\_function.h.



**16.81.2.2.14 volatile uint16\_t rxCount**

Number of bytes remaining to receive

Definition at line 137 of file `lpspi_shared_function.h`.

**16.81.2.2.15 uint8\_t rxDMAChannel**

Channel number for DMA rx channel

Definition at line 142 of file `lpspi_shared_function.h`.

**16.81.2.2.16 volatile uint16\_t rxFrameCnt**

Number of bytes from current frame which were already received

Definition at line 139 of file `lpspi_shared_function.h`.

**16.81.2.2.17 transfer\_status\_t status**

The status of the current

Definition at line 146 of file `lpspi_shared_function.h`.

**16.81.2.2.18 lpspi\_transfer\_type transferType**

Type of LPSPI transfer

Definition at line 144 of file `lpspi_shared_function.h`.

**16.81.2.2.19 const uint8\_t\* txBuff**

The buffer from which transmitted bytes are taken

Definition at line 134 of file `lpspi_shared_function.h`.

**16.81.2.2.20 volatile uint16\_t txCount**

Number of bytes remaining to send

Definition at line 136 of file `lpspi_shared_function.h`.

**16.81.2.2.21 uint8\_t txDMAChannel**

Channel number for DMA tx channel

Definition at line 143 of file `lpspi_shared_function.h`.

**16.81.2.2.22 volatile uint16\_t txFrameCnt**

Number of bytes from current frame which were already sent

Definition at line 138 of file `lpspi_shared_function.h`.

**16.81.2.3 struct lpspi\_slave\_config\_t**

User configuration structure for the SPI slave driver. Implements : `lpspi_slave_config_t_Class`.

Definition at line 47 of file `lpspi_slave_driver.h`.

**Data Fields**

- [lpspi\\_signal\\_polarity\\_t pcsPolarity](#)
- [uint16\\_t bitcount](#)
- [lpspi\\_clock\\_phase\\_t clkPhase](#)
- [lpspi\\_which\\_pcs\\_t whichPcs](#)
- [lpspi\\_sck\\_polarity\\_t clkPolarity](#)

- bool [lsbFirst](#)
- [lpspi\\_transfer\\_type](#) transferType
- uint8\_t [rxDMAChannel](#)
- uint8\_t [txDMAChannel](#)
- [spi\\_callback\\_t](#) callback
- void \* [callbackParam](#)

#### Field Documentation

##### 16.81.2.3.1 uint16\_t bitcount

Number of bits/frame, minimum is 8-bits

Definition at line 50 of file [lpspi\\_slave\\_driver.h](#).

##### 16.81.2.3.2 spi\_callback\_t callback

Select the callback to transfer complete

Definition at line 58 of file [lpspi\\_slave\\_driver.h](#).

##### 16.81.2.3.3 void\* callbackParam

Select additional callback parameters if it's necessary

Definition at line 59 of file [lpspi\\_slave\\_driver.h](#).

##### 16.81.2.3.4 lpspi\_clock\_phase\_t clkPhase

Selects which phase of clock to capture data

Definition at line 51 of file [lpspi\\_slave\\_driver.h](#).

##### 16.81.2.3.5 lpspi\_sck\_polarity\_t clkPolarity

Selects clock polarity

Definition at line 53 of file [lpspi\\_slave\\_driver.h](#).

##### 16.81.2.3.6 bool lsbFirst

Option to transmit LSB first

Definition at line 54 of file [lpspi\\_slave\\_driver.h](#).

##### 16.81.2.3.7 lpspi\_signal\_polarity\_t pcsPolarity

PCS polarity

Definition at line 49 of file [lpspi\\_slave\\_driver.h](#).

##### 16.81.2.3.8 uint8\_t rxDMAChannel

Channel number for DMA rx channel. If DMA mode isn't used this field will be ignored.

Definition at line 56 of file [lpspi\\_slave\\_driver.h](#).

##### 16.81.2.3.9 lpspi\_transfer\_type transferType

Type of LPSPi transfer

Definition at line 55 of file [lpspi\\_slave\\_driver.h](#).

##### 16.81.2.3.10 uint8\_t txDMAChannel

Channel number for DMA tx channel. If DMA mode isn't used this field will be ignored.

Definition at line 57 of file `lpspi_slave_driver.h`.

#### 16.81.2.3.11 `lpspi_which_pcs_t` whichPcs

Definition at line 52 of file `lpspi_slave_driver.h`.

### 16.81.3 Enumeration Type Documentation

#### 16.81.3.1 `enum lpspi_clock_phase_t`

LPSPI clock phase configuration. Implements : `lpspi_clock_phase_t_Class`.

Enumerator

***LPSPI\_CLOCK\_PHASE\_1ST\_EDGE*** Data captured on SCK 1st edge, changed on 2nd.

***LPSPI\_CLOCK\_PHASE\_2ND\_EDGE*** Data changed on SCK 1st edge, captured on 2nd.

Definition at line 80 of file `lpspi_shared_function.h`.

#### 16.81.3.2 `enum lpspi_sck_polarity_t`

LPSPI Clock Signal (SCK) Polarity configuration. Implements : `lpspi_sck_polarity_t_Class`.

Enumerator

***LPSPI\_SCK\_ACTIVE\_HIGH*** Signal is Active High (idles low).

***LPSPI\_SCK\_ACTIVE\_LOW*** Signal is Active Low (idles high).

Definition at line 89 of file `lpspi_shared_function.h`.

#### 16.81.3.3 `enum lpspi_signal_polarity_t`

LPSPI Signal (PCS and Host Request) Polarity configuration. Implements : `lpspi_signal_polarity_t_Class`.

Enumerator

***LPSPI\_ACTIVE\_HIGH*** Signal is Active High (idles low).

***LPSPI\_ACTIVE\_LOW*** Signal is Active Low (idles high).

Definition at line 71 of file `lpspi_shared_function.h`.

#### 16.81.3.4 `enum lpspi_transfer_type`

Type of LPSPI transfer (based on interrupts or DMA). Implements : `lpspi_transfer_type_Class`.

Enumerator

***LPSPI\_USING\_DMA*** The driver will use DMA to perform SPI transfer

***LPSPI\_USING\_INTERRUPTS*** The driver will use interrupts to perform SPI transfer

Definition at line 99 of file `lpspi_shared_function.h`.

#### 16.81.3.5 `enum lpspi_which_pcs_t`

LPSPI Peripheral Chip Select (PCS) configuration (which PCS to configure). Implements : `lpspi_which_pcs_t_Class`.

Enumerator

***LPSPI\_PCS0*** PCS[0]

***LPSPI\_PCS1*** PCS[1]

***LPSPI\_PCS2*** PCS[2]

***LPSPI\_PCS3*** PCS[3]

Definition at line 60 of file `lpspi_shared_function.h`.

#### 16.81.3.6 enum `transfer_status_t`

Type of error reported by LPSPI.

#### Enumerator

***LPSPI\_TRANSFER\_OK*** Transfer OK

***LPSPI\_TRANSMIT\_FAIL*** Error during transmission

***LPSPI\_RECEIVE\_FAIL*** Error during reception

Definition at line 107 of file `lpspi_shared_function.h`.

#### 16.81.4 Function Documentation

##### 16.81.4.1 void `LPSPi0_IRQHandler ( void )`

This function is the implementation of LPSPi0 handler named in startup code.

It passes the instance to the shared LPSPI IRQ handler.

Definition at line 115 of file `lpspi_irq.c`.

##### 16.81.4.2 void `LPSPi1_IRQHandler ( void )`

This function is the implementation of LPSPi1 handler named in startup code.

It passes the instance to the shared LPSPI IRQ handler.

Definition at line 127 of file `lpspi_irq.c`.

##### 16.81.4.3 void `LPSPi2_IRQHandler ( void )`

This function is the implementation of LPSPi2 handler named in startup code.

It passes the instance to the shared LPSPI IRQ handler.

Definition at line 139 of file `lpspi_irq.c`.

##### 16.81.4.4 void `LPSPi_DRV_DisableTEIEInterrupts ( uint32_t instance )`

Disable the TEIE interrupts at the end of a transfer. Disable the interrupts and clear the status for transmit/receive errors.

Definition at line 235 of file `lpspi_shared_function.c`.

##### 16.81.4.5 void `LPSPi_DRV_FillupTxBuffer ( uint32_t instance )`

The function `LPSPi_DRV_FillupTxBuffer` writes data in TX hardware buffer depending on driver state and number of bytes remained to send.

The function `LPSPi_DRV_FillupTxBuffer` writes data in TX hardware buffer depending on driver state and number of bytes remained to send.

Definition at line 123 of file `lpspi_shared_function.c`.

#### 16.81.4.6 void LPSPI\_DRV\_IRQHandler ( uint32\_t instance )

The function LPSPI\_DRV\_IRQHandler passes IRQ control to either the master or slave driver.

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.

Definition at line 100 of file lpspi\_shared\_function.c.

#### 16.81.4.7 status\_t LPSPI\_DRV\_MasterAbortTransfer ( uint32\_t instance )

Terminates an interrupt driven asynchronous transfer early.

During an a-sync (non-blocking) 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 LPSPI peripheral.
-----------------	--

##### Returns

STATUS\_SUCCESS The transfer was successful, or LPSPI\_STATUS\_NO\_TRANSFER\_IN\_PROGRESS No transfer is currently in progress.

Definition at line 578 of file lpspi\_master\_driver.c.

#### 16.81.4.8 status\_t LPSPI\_DRV\_MasterConfigureBus ( uint32\_t instance, const lpspi\_master\_config\_t \* spiConfig, uint32\_t \* calculatedBaudRate )

Configures the LPSPI port physical parameters to access a device on the bus when the LSPI instance is configured for interrupt operation.

In this function, the term "spiConfig" is used to indicate the SPI device for which the LPSPI master is communicating. This is an optional function as the spiConfig parameters are normally configured in the initialization function or the transfer functions, where these various functions would call the configure bus function. This is an example to set up the [lpspi\\_master\\_config\\_t](#) structure to call the LPSPI\_DRV\_MasterConfigureBus function by passing in these parameters:

```
1 lpspi_master_config_t spiConfig1;  You can also declare spiConfig2, spiConfig3, etc
2 spiConfig1.bitsPerSec = 500000;
3 spiConfig1.whichPcs = LPSPI_PCS0;
4 spiConfig1.pcsPolarity = LPSPI_ACTIVE_LOW;
5 spiConfig1.isPcsContinuous = false;
6 spiConfig1.bitCount = 16;
7 spiConfig1.clkPhase = LPSPI_CLOCK_PHASE_1ST_EDGE;
8 spiConfig1.clkPolarity = LPSPI_ACTIVE_HIGH;
9 spiConfig1.lsbFirst= false;
10 spiConfig.transferType = LPSPI_USING_INTERRUPTS;
```

##### Parameters

<i>instance</i>	The instance number of the LPSPI peripheral.
<i>spiConfig</i>	Pointer to the spiConfig structure. This structure contains the settings for the SPI bus configuration. The SPI device parameters are the desired baud rate (in bits-per-sec), bits-per-frame, chip select attributes, clock attributes, 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**

STATUS\_SUCCESS The transfer has completed successfully, or STATUS\_ERROR if driver is error and needs to clean error.

Definition at line 309 of file lpspi\_master\_driver.c.

#### 16.81.4.9 status\_t LPSPI\_DRV\_MasterDeinit ( uint32\_t instance )

Shuts down a LPSPI instance.

This function resets the LPSPI peripheral, gates its clock, and disables the interrupt to the core. It first checks to see if a transfer is in progress and if so returns an error status.

**Parameters**

<i>instance</i>	The instance number of the LPSPI peripheral.
-----------------	--

**Returns**

STATUS\_SUCCESS The transfer has completed successfully, or STATUS\_BUSY The transfer is still in progress. STATUS\_ERROR if driver is error and needs to clean error.

Definition at line 219 of file lpspi\_master\_driver.c.

#### 16.81.4.10 void LPSPI\_DRV\_MasterGetDefaultConfig ( lpspi\_master\_config\_t \* spiConfig )

Return default configuration for SPI master.

Initializes a structured provided by user with the configuration of an interrupt based LPSPI transfer. Source clock for LPSPI is configured to 8MHz. If the applications uses other frequency is necessary to update lpspiSrcClk field.

**Parameters**

<i>spiConfig</i>	Pointer to configuration structure which is filled with default configuration
------------------	---

Definition at line 131 of file lpspi\_master\_driver.c.

#### 16.81.4.11 status\_t LPSPI\_DRV\_MasterGetTransferStatus ( uint32\_t instance, uint32\_t \* bytesRemained )

Returns whether the previous interrupt driven transfer is completed.

When performing an a-sync (non-blocking) 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 LPSPI peripheral.
<i>bytesRemained</i>	Pointer to a value that is filled in with the number of bytes that must be received.

**Returns**

STATUS\_SUCCESS The transfer has completed successfully, or STATUS\_BUSY The transfer is still in progress. framesTransferred is filled with the number of words that have been transferred so far.

Definition at line 549 of file lpspi\_master\_driver.c.

#### 16.81.4.12 status\_t LPSPI\_DRV\_MasterInit ( uint32\_t instance, lpspi\_state\_t \* lpspiState, const lpspi\_master\_config\_t \* spiConfig )

Initializes a LPSPI instance for interrupt driven master mode operation.

This function uses an interrupt-driven method for transferring data. In this function, the term "spiConfig" is used to indicate the SPI device for which the LPSPI master is communicating. This function initializes the run-time state structure to track the ongoing transfers, un-gates the clock to the LPSPI module, resets the LPSPI module, configures the IRQ state structure, enables the module-level interrupt to the core, and enables the LPSPI module. This is an example to set up the `lpspi_master_state_t` and call the `LPSPI_DRV_MasterInit` function by passing in these parameters:

```
1 lpspi_master_state_t lpspiMasterState; <- the user allocates memory for this structure
2 lpspi_master_config_t spiConfig; Can declare more configs for use in transfer functions
3 spiConfig.bitsPerSec = 500000;
4 spiConfig.whichPcs = LPSPI_PCS0;
5 spiConfig.pcsPolarity = LPSPI_ACTIVE_LOW;
6 spiConfig.isPcsContinuous = false;
7 spiConfig.bitCount = 16;
8 spiConfig.clkPhase = LPSPI_CLOCK_PHASE_1ST_EDGE;
9 spiConfig.clkPolarity = LPSPI_ACTIVE_HIGH;
10 spiConfig.lsbFirst= false;
11 spiConfig.transferType = LPSPI_USING_INTERRUPTS;
12 LPSPI_DRV_MasterInit(masterInstance, &lpspiMasterState, &spiConfig);
```

#### Parameters

<i>instance</i>	The instance number of the LPSPI peripheral.
<i>lpspiState</i>	The pointer to the LPSPI master driver state structure. The user passes the memory for this run-time state structure. The LPSPI master driver populates the members. This run-time state structure keeps track of the transfer in progress.
<i>spiConfig</i>	The data structure containing information about a device on the SPI bus

#### Returns

An error code or `STATUS_SUCCESS`.

Definition at line 166 of file `lpspi_master_driver.c`.

#### 16.81.4.13 void LPSPI\_DRV\_MasterIRQHandler ( uint32\_t instance )

Interrupt handler for LPSPI master mode. This handler uses the buffers stored in the `lpspi_master_state_t` structs to transfer data.

#### Parameters

<i>instance</i>	The instance number of the LPSPI peripheral.
-----------------	--

Interrupt handler for LPSPI master mode. This handler uses the buffers stored in the `lpspi_master_state_t` structs to transfer data.

Definition at line 867 of file `lpspi_master_driver.c`.

#### 16.81.4.14 status\_t LPSPI\_DRV\_MasterSetDelay ( uint32\_t instance, uint32\_t delayBetweenTransfers, uint32\_t delaySCKtoPCS, uint32\_t delayPCStoSCK )

Configures the LPSPI master mode bus timing delay options.

This function involves the LPSPI 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 LPSPI module has been initialized for master mode. The timings are adjusted in terms of cycles of the baud rate clock. The bus timing delays that can be adjusted are listed below:

**SCK to PCS Delay:** Adjustable delay option between the last edge of SCK to the de-assertion of the PCS signal.

**PCS to SCK Delay:** Adjustable delay option between the assertion of the PCS signal to the first SCK edge.

**Delay between Transfers:** 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.

**Parameters**

<i>instance</i>	The instance number of the LPSPI peripheral.
<i>delayBetween↔ Transfers</i>	Minimum delay between 2 transfers in clock cycles
<i>delaySCKtoP↔ CS</i>	Minimum delay between SCK and PCS in clock cycles
<i>delayPCStoS↔ CK</i>	Minimum delay between PCS and SCK in clock cycles

**Returns**

STATUS\_SUCCESS The transfer has completed successfully, or STATUS\_ERROR if driver is error and needs to clean error.

Definition at line 266 of file lpspi\_master\_driver.c.

**16.81.4.15** `status_t LPSPI_DRV_MasterTransfer ( uint32_t instance, const uint8_t * sendBuffer, uint8_t * receiveBuffer, uint16_t transferByteCount )`

Performs an interrupt driven non-blocking SPI master mode transfer.

This function simultaneously sends and receives data on the SPI bus, as SPI is naturally a full-duplex bus. The function returns immediately after initiating the transfer. The user needs to check whether the transfer is complete using the LPSPI\_DRV\_MasterGetTransferStatus function. This function allows the user to optionally pass in a SPI configuration structure which allows the user to change the SPI bus attributes in conjunction with initiating a SPI transfer. The difference between passing in the SPI configuration structure here as opposed to the configure bus function is that the configure bus function returns the calculated baud rate where this function does not. The user can also call the configure bus function prior to the transfer in which case the user would simply pass in a NULL to the transfer function's device structure parameter. Depending on frame size sendBuffer and receiveBuffer must be aligned like this: -1 byte if frame size ≤ 8 bits -2 bytes if 8 bits < frame size ≤ 16 bits -4 bytes if 16 bits < frame size

**Parameters**

<i>instance</i>	The instance number of the LPSPI peripheral.
<i>spiConfig</i>	Pointer to the SPI configuration 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 LPSPI_DRV_MasterConfigureBus function.
<i>sendBuffer</i>	The pointer to the data buffer of the data to send. You may pass NULL for this parameter and bytes with a value of 0 (zero) is sent.
<i>receiveBuffer</i>	Pointer to the buffer where the received bytes are stored. If you pass NULL for this parameter, the received bytes are ignored.
<i>transferByte↔ Count</i>	The number of bytes to send and receive which is equal to size of send or receive buffers

**Returns**

STATUS\_SUCCESS The transfer was successful, or STATUS\_BUSY Cannot perform transfer because a transfer is already in progress

Definition at line 511 of file lpspi\_master\_driver.c.

**16.81.4.16** `status_t LPSPI_DRV_MasterTransferBlocking ( uint32_t instance, const uint8_t * sendBuffer, uint8_t * receiveBuffer, uint16_t transferByteCount, uint32_t timeout )`

Performs an interrupt driven blocking SPI master mode transfer.

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. This function allows the user to optionally pass in a SPI



configuration structure which allows the user to change the SPI bus attributes in conjunction with initiating a SPI transfer. The difference between passing in the SPI configuration structure here as opposed to the configure bus function is that the configure bus function returns the calculated baud rate where this function does not. The user can also call the configure bus function prior to the transfer in which case the user would simply pass in a NULL to the transfer function's device structure parameter. Depending on frame size sendBuffer and receiveBuffer must be aligned like this: -1 byte if frame size <= 8 bits -2 bytes if 8 bits < frame size <= 16 bits -4 bytes if 16 bits < frame size

#### Parameters

<i>instance</i>	The instance number of the LPSPI peripheral.
<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 which is equal to size of send or receive buffers
<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 STATUS_TIMEOUT error returned.

#### Returns

STATUS\_SUCCESS The transfer was successful, or STATUS\_BUSY Cannot perform transfer because a transfer is already in progress, or STATUS\_TIMEOUT The transfer timed out and was aborted.

Definition at line 424 of file lpspi\_master\_driver.c.

#### 16.81.4.17 void LPSPI\_DRV\_ReadRXBuffer ( uint32\_t instance )

The function LPSPI\_DRV\_ReadRXBuffer reads data from RX hardware buffer and writes this data in RX software buffer.

The function LPSPI\_DRV\_ReadRXBuffer reads data from RX hardware buffer and writes this data in RX software buffer.

Definition at line 192 of file lpspi\_shared\_function.c.

#### 16.81.4.18 status\_t LPSPI\_DRV\_SetPcs ( uint32\_t instance, lpspi\_which\_pcs\_t whichPcs, lpspi\_signal\_polarity\_t polarity )

Select the chip to communicate with.

The main purpose of this function is to set the PCS and the appropriate polarity.

#### Parameters

<i>instance</i>	LPSPI instance
<i>whichPcs</i>	selected chip
<i>polarity</i>	chip select line polarity

#### Returns

STATUS\_SUCCESS The transfer has completed successfully, or STATUS\_ERROR if driver is error and needs to clean error.

Definition at line 600 of file lpspi\_master\_driver.c.

#### 16.81.4.19 status\_t LPSPI\_DRV\_SlaveAbortTransfer ( uint32\_t instance )

Aborts the transfer that started by a non-blocking call transfer function.

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**

STATUS\_SUCCESS if everything is OK.

Definition at line 512 of file `lpspi_slave_driver.c`.

#### 16.81.4.20 `status_t LPSPI_DRV_SlaveDeinit ( uint32_t instance )`

Shuts down an LPSPI instance interrupt mechanism.

Disables the LPSPI module, gates its clock, and changes the LPSPI slave driver state to NonInit for the LPSPI slave module which is initialized with interrupt mechanism. After de-initialization, the user can re-initialize the LPSPI slave module with other mechanisms.

**Parameters**

<i>instance</i>	The instance number of the LPSPI peripheral.
-----------------	--

**Returns**

STATUS\_SUCCESS if driver starts to send/receive data successfully. STATUS\_ERROR if driver is error and needs to clean error. STATUS\_BUSY if a transfer is in progress

Definition at line 201 of file `lpspi_slave_driver.c`.

#### 16.81.4.21 `void LPSPI_DRV_SlaveGetDefaultConfig ( lpspi_slave_config_t * spiConfig )`

Return default configuration for SPI master.

Initializes a structured provided by user with the configuration of an interrupt based LPSPI transfer.

Definition at line 105 of file `lpspi_slave_driver.c`.

#### 16.81.4.22 `status_t LPSPI_DRV_SlaveGetTransferStatus ( uint32_t instance, uint32_t * bytesRemained )`

Returns whether the previous transfer is finished.

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 bytes that have been transferred up to now.

**Parameters**

<i>instance</i>	The instance number of the LPSPI peripheral.
<i>bytesRemained</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**

STATUS\_SUCCESS The transfer has completed successfully, or STATUS\_BUSY The transfer is still in progress. STATUS\_ERROR if driver is error and needs to clean error.

Definition at line 550 of file `lpspi_slave_driver.c`.

#### 16.81.4.23 `status_t LPSPI_DRV_SlaveInit ( uint32_t instance, lpspi_state_t * lpspiState, const lpspi_slave_config_t * slaveConfig )`

Initializes a LPSPI instance for a slave mode operation, using interrupt mechanism.

This function un-gates the clock to the LPSPI module, initializes the LPSPI for slave mode. After it is initialized, the LPSPI 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 LPSPI slave uses an interrupt mechanism.

## Parameters

<i>instance</i>	The instance number of the LPSPI peripheral.
<i>lpspiState</i>	The pointer to the LPSPI slave driver state structure.
<i>slaveConfig</i>	The configuration structure <code>lpspi_slave_user_config_t</code> which configures the data bus format.

## Returns

An error code or STATUS\_SUCCESS.

Definition at line 125 of file `lpspi_slave_driver.c`.

## 16.81.4.24 void LPSPI\_DRV\_SlaveIRQHandler ( uint32\_t instance )

Interrupt handler for LPSPI slave mode. This handler uses the buffers stored in the `lpspi_master_state_t` structs to transfer data.

## Parameters

<i>instance</i>	The instance number of the LPSPI peripheral.
-----------------	--

Definition at line 437 of file `lpspi_slave_driver.c`.

## 16.81.4.25 status\_t LPSPI\_DRV\_SlaveTransfer ( uint32\_t instance, const uint8\_t \* sendBuffer, uint8\_t \* receiveBuffer, uint16\_t transferByteCount )

Starts the transfer data on LPSPI bus using a non-blocking call.

This function checks the driver status and mechanism and transmits/receives data through the LPSPI 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. Depending on frame size `sendBuffer` and `receiveBuffer` must be aligned like this: -1 byte if frame size  $\leq 8$  bits -2 bytes if  $8 \text{ bits} < \text{frame size} \leq 16$  bits -4 bytes if  $16 \text{ bits} < \text{frame size}$

## Parameters

<i>instance</i>	The instance number of LPSPI peripheral
<i>sendBuffer</i>	The pointer to data that user wants to transmit.
<i>receiveBuffer</i>	The pointer to data that user wants to store received data.
<i>transferByteCount</i>	The number of bytes to send and receive which is equal to size of send or receive buffers

## Returns

STATUS\_SUCCESS if driver starts to send/receive data successfully. STATUS\_ERROR if driver is error and needs to clean error. STATUS\_BUSY if a transfer is in progress

Definition at line 280 of file `lpspi_slave_driver.c`.

## 16.81.4.26 status\_t LPSPI\_DRV\_SlaveTransferBlocking ( uint32\_t instance, const uint8\_t \* sendBuffer, uint8\_t \* receiveBuffer, uint16\_t transferByteCount, uint32\_t timeout )

Transfers data on LPSPI bus using a blocking call.

This function checks the driver status and mechanism and transmits/receives data through the LPSPI 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. Depending on frame size `sendBuffer` and `receiveBuffer` must be aligned like this: -1 byte if frame size  $\leq 8$  bits -2 bytes if  $8 \text{ bits} < \text{frame size} \leq 16$  bits -4 bytes if  $16 \text{ bits} < \text{frame size}$

**Parameters**

<i>instance</i>	The instance number of LPSPI peripheral
<i>sendBuffer</i>	The pointer to data that user wants to transmit.
<i>receiveBuffer</i>	The pointer to data that user wants to store received data.
<i>transferByteCount</i>	The number of bytes to send and receive which is equal to size of send or receive buffers
<i>timeout</i>	The maximum number of milliseconds that function waits before timed out reached.

**Returns**

STATUS\_SUCCESS if driver starts to send/receive data successfully. STATUS\_ERROR if driver is error and needs to clean error. STATUS\_BUSY if a transfer is in progress STATUS\_TIMEOUT if time out reached while transferring is in progress.

Definition at line 230 of file `lpspi_slave_driver.c`.

**16.81.5 Variable Documentation****16.81.5.1 LPSPI\_Type\* g\_lpspiBase[LPSPI\_INSTANCE\_COUNT]**

Table of base pointers for SPI instances.

Definition at line 78 of file `lpspi_shared_function.c`.

**16.81.5.2 IRQn\_Type g\_lpspiIrqlId[LPSPI\_INSTANCE\_COUNT]**

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

Definition at line 81 of file `lpspi_shared_function.c`.

**16.81.5.3 lpspi\_state\_t\* g\_lpspiStatePtr[LPSPI\_INSTANCE\_COUNT]**

Definition at line 84 of file `lpspi_shared_function.c`.

## 16.82 LPTMR Driver

### 16.82.1 Detailed Description

Low Power Timer Peripheral Driver.

The LPTMR is a configurable general-purpose 16-bit counter that has two operational modes: Timer and Pulse-Counter.

Depending on the configured operational mode, the counter in the LPTMR can be incremented using a clock input (Timer mode) or an event counter (external events like button presses or internal events from different trigger sources).

#### Timer Mode

In Timer mode, the LPTMR increments the internal counter from a selectable clock source. An optional 16-bit prescaler can be configured.

#### Pulse-Counter Mode

In Pulse-Counter Mode, the LPTMR counter increments from a selectable trigger source, input pin, which can be an external event (like a button press) or internal events (like triggers from TRGMUX).

An optional 16-bit glitch-filter can be configured to reject events that have a duration below a set period.

#### Initialization prerequisites

Before configuring the LPTMR, the peripheral clock must be enabled from the PCC module.

The peripheral clock must not be confused with the counter clock, which is selectable within the LPTMR.

#### Driver configuration

The LPTMR driver allows configuring the LPTMR for Pulse-Counter Mode or Timer Mode via the general configuration structure.

Configurable options:

- work mode (timer or pulse-counter)
- enable interrupts and DMA requests
- free running mode (overflow mode of the counter)
- compare value (interrupt generation on counter value)
- compare value measurement units (counter ticks or microseconds)
- input clock selection
- prescaler/glitch filter configuration
- enable bypass prescaler
- pin select (for pulse-counter mode)
- input pin and polarity (for pulse-counter mode)

```
/* LPTMR initialization of config structure */
lptmr_config_t config = {
    .workMode = LPTMR_WORKMODE_TIMER,
    .dmaRequest = false,
    .interruptEnable = false,
    .freeRun = false,
    .compareValue = 1000U,
    .counterUnits = LPTMR_COUNTER_UNITS_TICKS,
    .clockSelect = LPTMR_CLOCKSOURCE_SIRCDIV2,
    .prescaler = LPTMR_PRESCALE_2,
    .bypassPrescaler = false,
```

```

        .pinSelect = LPTMR_PINSELECT_TRGMUX,
        .pinPolarity = LPTMR_PINPOLARITY_RISING,
    };

    /* Initialize the LPTMR and start the counter in a separate operation */
    status = LPTMR_DRV_Init(0, &config, false);
    /* Start timer counting */
    LPTMR_DRV_StartCounter(0);

```

## API

Some of the features exposed by the API are targeted specifically for Timer Mode or Pulse-Counter Mode. For example, configuring the Compare Value in microseconds makes sense only for Timer Mode, so therefore it is restricted for use in Pulse-Counter mode.

For any invalid configuration the functions will either return an error code or trigger DEV\_ASSERT (if enabled). For more details, please refer to each function description.

## Integration guideline

### Compilation units

The following files need to be compiled in the project:

```

${S32SDK_PATH}\platform\drivers\src\lptmr_driver.c
${S32SDK_PATH}\platform\drivers\src\lptmr_hw_access.c

```

### Include path

The following paths need to be added to the include path of the toolchain:

```

${S32SDK_PATH}\platform\drivers\inc\

```

### Compile symbols

No special symbols are required for this component

### Dependencies

### Clock Manager

## Data Structures

- struct [lptmr\\_config\\_t](#)  
*Defines the configuration structure for LPTMR. [More...](#)*

## Enumerations

- enum [lptmr\\_pinselect\\_t](#) { [LPTMR\\_PINSELECT\\_TRGMUX](#) = 0x00u, [LPTMR\\_PINSELECT\\_ALT2](#) = 0x02u, [LPTMR\\_PINSELECT\\_ALT3](#) = 0x03u }  
*Pulse Counter Input selection Implements : [lptmr\\_pinselect\\_t\\_Class](#).*
- enum [lptmr\\_pinpolarity\\_t](#) { [LPTMR\\_PINPOLARITY\\_RISING](#) = 0u, [LPTMR\\_PINPOLARITY\\_FALLING](#) = 1u }  
*Pulse Counter input polarity Implements : [lptmr\\_pinpolarity\\_t\\_Class](#).*
- enum [lptmr\\_workmode\\_t](#) { [LPTMR\\_WORKMODE\\_TIMER](#) = 0u, [LPTMR\\_WORKMODE\\_PULSECOUNTER](#) = 1u }  
*Work Mode Implements : [lptmr\\_workmode\\_t\\_Class](#).*
- enum [lptmr\\_prescaler\\_t](#) {  
[LPTMR\\_PRESCALE\\_2](#) = 0x00u, [LPTMR\\_PRESCALE\\_4\\_GLITCHFILTER\\_2](#) = 0x01u, [LPTMR\\_PRESCALE\\_4\\_GLITCHFILTER\\_4](#) = 0x02u, [LPTMR\\_PRESCALE\\_16\\_GLITCHFILTER\\_8](#) = 0x03u,  
[LPTMR\\_PRESCALE\\_32\\_GLITCHFILTER\\_16](#) = 0x04u, [LPTMR\\_PRESCALE\\_64\\_GLITCHFILTER\\_32](#) =

```
0x05u, LPTMR_PRESCALE_128_GLITCHFILTER_64 = 0x06u, LPTMR_PRESCALE_256_GLITCHFILTER_128 = 0x07u,
LPTMR_PRESCALE_512_GLITCHFILTER_256 = 0x08u, LPTMR_PRESCALE_1024_GLITCHFILTER_512 = 0x09u, LPTMR_PRESCALE_2048_GLITCHFILTER_1024 = 0x0Au, LPTMR_PRESCALE_4096_GLITCHFILTER_2048 = 0x0Bu,
LPTMR_PRESCALE_8192_GLITCHFILTER_4096 = 0x0Cu, LPTMR_PRESCALE_16384_GLITCHFILTER_8192 = 0x0Du, LPTMR_PRESCALE_32768_GLITCHFILTER_16384 = 0x0Eu, LPTMR_PRESCALE_65536_GLITCHFILTER_32768 = 0x0Fu }
```

*Prescaler Selection Implements : `lptmr_prescaler_t` Class.*

- enum `lptmr_clocksource_t` { `LPTMR_CLOCKSOURCE_SIRCDIV2` = 0x00u, `LPTMR_CLOCKSOURCE_1_KHZ_LPO` = 0x01u, `LPTMR_CLOCKSOURCE_RTC` = 0x02u, `LPTMR_CLOCKSOURCE_PCC` = 0x03u }

*Clock Source selection Implements : `lptmr_clocksource_t` Class.*

- enum `lptmr_counter_units_t` { `LPTMR_COUNTER_UNITS_TICKS` = 0x00U, `LPTMR_COUNTER_UNITS_MICROSECONDS` = 0x01U }

*Defines the LPTMR counter units available for configuring or reading the timer compare value.*

### LPTMR Driver Functions

- void `LPTMR_DRV_InitConfigStruct` (`lptmr_config_t` \*const config)  
*Initialize a configuration structure with default values.*
- void `LPTMR_DRV_Init` (const uint32\_t instance, const `lptmr_config_t` \*const config, const bool startCounter)  
*Initialize a LPTMR instance with values from an input configuration structure.*
- void `LPTMR_DRV_SetConfig` (const uint32\_t instance, const `lptmr_config_t` \*const config)  
*Configure a LPTMR instance.*
- void `LPTMR_DRV_GetConfig` (const uint32\_t instance, `lptmr_config_t` \*const config)  
*Get the current configuration of a LPTMR instance.*
- void `LPTMR_DRV_Deinit` (const uint32\_t instance)  
*De-initialize a LPTMR instance.*
- status\_t `LPTMR_DRV_SetCompareValueByCount` (const uint32\_t instance, const uint16\_t compareValueByCount)  
*Set the compare value in counter tick units, for a LPTMR instance.*
- void `LPTMR_DRV_GetCompareValueByCount` (const uint32\_t instance, uint16\_t \*const compareValueByCount)  
*Get the compare value in counter tick units, of a LPTMR instance.*
- status\_t `LPTMR_DRV_SetCompareValueByUs` (const uint32\_t instance, const uint32\_t compareValueUs)  
*Set the compare value for Timer Mode in microseconds, for a LPTMR instance.*
- void `LPTMR_DRV_GetCompareValueByUs` (const uint32\_t instance, uint32\_t \*const compareValueUs)  
*Get the compare value in microseconds, of a LPTMR instance.*
- bool `LPTMR_DRV_GetCompareFlag` (const uint32\_t instance)  
*Get the current state of the Compare Flag of a LPTMR instance.*
- void `LPTMR_DRV_ClearCompareFlag` (const uint32\_t instance)  
*Clear the Compare Flag of a LPTMR instance.*
- bool `LPTMR_DRV_IsRunning` (const uint32\_t instance)  
*Get the run state of a LPTMR instance.*
- void `LPTMR_DRV_SetInterrupt` (const uint32\_t instance, const bool enableInterrupt)  
*Enable/disable the LPTMR interrupt.*
- uint16\_t `LPTMR_DRV_GetCounterValueByCount` (const uint32\_t instance)  
*Get the current counter value in counter tick units.*
- void `LPTMR_DRV_StartCounter` (const uint32\_t instance)  
*Enable the LPTMR / Start the counter.*
- void `LPTMR_DRV_StopCounter` (const uint32\_t instance)  
*Disable the LPTMR / Stop the counter.*

- void [LPTMR\\_DRV\\_SetPinConfiguration](#) (const uint32\_t instance, const [lptmr\\_pinselect\\_t](#) pinSelect, const [lptmr\\_pinpolarity\\_t](#) pinPolarity)

*Set the Input Pin configuration for Pulse Counter mode.*

## 16.82.2 Data Structure Documentation

### 16.82.2.1 struct [lptmr\\_config\\_t](#)

Defines the configuration structure for LPTMR.

Implements : [lptmr\\_config\\_t\\_Class](#)

Definition at line 110 of file [lptmr\\_driver.h](#).

#### Data Fields

- bool [dmaRequest](#)
- bool [interruptEnable](#)
- bool [freeRun](#)
- [lptmr\\_workmode\\_t](#) workMode
- [lptmr\\_clocksource\\_t](#) clockSelect
- [lptmr\\_prescaler\\_t](#) prescaler
- bool [bypassPrescaler](#)
- uint32\_t [compareValue](#)
- [lptmr\\_counter\\_units\\_t](#) counterUnits
- [lptmr\\_pinselect\\_t](#) pinSelect
- [lptmr\\_pinpolarity\\_t](#) pinPolarity

#### Field Documentation

##### 16.82.2.1.1 bool [bypassPrescaler](#)

Enable/Disable prescaler bypass

Definition at line 120 of file [lptmr\\_driver.h](#).

##### 16.82.2.1.2 [lptmr\\_clocksource\\_t](#) [clockSelect](#)

Clock selection for Timer/Glitch filter

Definition at line 118 of file [lptmr\\_driver.h](#).

##### 16.82.2.1.3 uint32\_t [compareValue](#)

Compare value

Definition at line 121 of file [lptmr\\_driver.h](#).

##### 16.82.2.1.4 [lptmr\\_counter\\_units\\_t](#) [counterUnits](#)

Compare value units

Definition at line 122 of file [lptmr\\_driver.h](#).

##### 16.82.2.1.5 bool [dmaRequest](#)

Enable/Disable DMA requests

Definition at line 113 of file [lptmr\\_driver.h](#).



## 16.82.2.1.6 bool freeRun

Enable/Disable Free Running Mode

Definition at line 115 of file lptmr\_driver.h.

## 16.82.2.1.7 bool interruptEnable

Enable/Disable Interrupt

Definition at line 114 of file lptmr\_driver.h.

## 16.82.2.1.8 lptmr\_pinpolarity\_t pinPolarity

Pin Polarity for Pulse-Counter

Definition at line 125 of file lptmr\_driver.h.

## 16.82.2.1.9 lptmr\_pinselect\_t pinSelect

Pin selection for Pulse-Counter

Definition at line 124 of file lptmr\_driver.h.

## 16.82.2.1.10 lptmr\_prescaler\_t prescaler

Prescaler Selection

Definition at line 119 of file lptmr\_driver.h.

## 16.82.2.1.11 lptmr\_workmode\_t workMode

Time/Pulse Counter Mode

Definition at line 116 of file lptmr\_driver.h.

## 16.82.3 Enumeration Type Documentation

## 16.82.3.1 enum lptmr\_clocksource\_t

Clock Source selection Implements : lptmr\_clocksource\_t\_Class.

Enumerator

**LPTMR\_CLOCKSOURCE\_SIRCDIV2** SIRC clock  
**LPTMR\_CLOCKSOURCE\_1KHZ\_LPO** 1kHz LPO clock  
**LPTMR\_CLOCKSOURCE\_RTC** RTC clock  
**LPTMR\_CLOCKSOURCE\_PCC** PCC configured clock

Definition at line 87 of file lptmr\_driver.h.

## 16.82.3.2 enum lptmr\_counter\_units\_t

Defines the LPTMR counter units available for configuring or reading the timer compare value.

Implements : lptmr\_counter\_units\_t\_Class

Enumerator

**LPTMR\_COUNTER\_UNITS\_TICKS**  
**LPTMR\_COUNTER\_UNITS\_MICROSECONDS**

Definition at line 99 of file lptmr\_driver.h.

### 16.82.3.3 enum `lptmr_pinpolarity_t`

Pulse Counter input polarity Implements : `lptmr_pinpolarity_t_Class`.

Enumerator

**`LPTMR_PINPOLARITY_RISING`** Count pulse on rising edge

**`LPTMR_PINPOLARITY_FALLING`** Count pulse on falling edge

Definition at line 49 of file `lptmr_driver.h`.

### 16.82.3.4 enum `lptmr_pinselect_t`

Pulse Counter Input selection Implements : `lptmr_pinselect_t_Class`.

Enumerator

**`LPTMR_PINSELECT_TRGMUX`** Count pulses from TRGMUX trigger

**`LPTMR_PINSELECT_ALT2`** Count pulses from pin alternative 2

**`LPTMR_PINSELECT_ALT3`** Count pulses from pin alternative 3

Definition at line 37 of file `lptmr_driver.h`.

### 16.82.3.5 enum `lptmr_prescaler_t`

Prescaler Selection Implements : `lptmr_prescaler_t_Class`.

Enumerator

**`LPTMR_PRESCALE_2`** Timer mode: prescaler 2, Glitch filter mode: invalid

**`LPTMR_PRESCALE_4_GLITCHFILTER_2`** Timer mode: prescaler 4, Glitch filter mode: 2 clocks

**`LPTMR_PRESCALE_8_GLITCHFILTER_4`** Timer mode: prescaler 8, Glitch filter mode: 4 clocks

**`LPTMR_PRESCALE_16_GLITCHFILTER_8`** Timer mode: prescaler 16, Glitch filter mode: 8 clocks

**`LPTMR_PRESCALE_32_GLITCHFILTER_16`** Timer mode: prescaler 32, Glitch filter mode: 16 clocks

**`LPTMR_PRESCALE_64_GLITCHFILTER_32`** Timer mode: prescaler 64, Glitch filter mode: 32 clocks

**`LPTMR_PRESCALE_128_GLITCHFILTER_64`** Timer mode: prescaler 128, Glitch filter mode: 64 clocks

**`LPTMR_PRESCALE_256_GLITCHFILTER_128`** Timer mode: prescaler 256, Glitch filter mode: 128 clocks

**`LPTMR_PRESCALE_512_GLITCHFILTER_256`** Timer mode: prescaler 512, Glitch filter mode: 256 clocks

**`LPTMR_PRESCALE_1024_GLITCHFILTER_512`** Timer mode: prescaler 1024, Glitch filter mode: 512 clocks

**`LPTMR_PRESCALE_2048_GLITCHFILTER_1024`** Timer mode: prescaler 2048, Glitch filter mode: 1024 clocks

**`LPTMR_PRESCALE_4096_GLITCHFILTER_2048`** Timer mode: prescaler 4096, Glitch filter mode: 2048 clocks

**`LPTMR_PRESCALE_8192_GLITCHFILTER_4096`** Timer mode: prescaler 8192, Glitch filter mode: 4096 clocks

**`LPTMR_PRESCALE_16384_GLITCHFILTER_8192`** Timer mode: prescaler 16384, Glitch filter mode: 8192 clocks

**`LPTMR_PRESCALE_32768_GLITCHFILTER_16384`** Timer mode: prescaler 32768, Glitch filter mode: 16384 clocks

**`LPTMR_PRESCALE_65536_GLITCHFILTER_32768`** Timer mode: prescaler 65536, Glitch filter mode: 32768 clocks

Definition at line 65 of file `lptmr_driver.h`.

16.82.3.6 enum `lptmr_workmode_t`

Work Mode Implements : `lptmr_workmode_t_Class`.

## Enumerator

**`LPTMR_WORKMODE_TIMER`** Timer

**`LPTMR_WORKMODE_PULSECOUNTER`** Pulse counter

Definition at line 57 of file `lptmr_driver.h`.

## 16.82.4 Function Documentation

16.82.4.1 void `LPTMR_DRV_ClearCompareFlag ( const uint32_t instance )`

Clear the Compare Flag of a LPTMR instance.

## Parameters

in	<i>instance</i>	- LPTMR instance number
----	-----------------	-------------------------

16.82.4.2 void `LPTMR_DRV_Deinit ( const uint32_t instance )`

De-initialize a LPTMR instance.

## Parameters

in	<i>instance</i>	- LPTMR instance number
----	-----------------	-------------------------

16.82.4.3 bool `LPTMR_DRV_GetCompareFlag ( const uint32_t instance )`

Get the current state of the Compare Flag of a LPTMR instance.

## Parameters

in	<i>instance</i>	- LPTMR instance number
----	-----------------	-------------------------

## Returns

The state of the Compare Flag

16.82.4.4 void `LPTMR_DRV_GetCompareValueByCount ( const uint32_t instance, uint16_t *const compareValueByCount )`

Get the compare value in counter tick units, of a LPTMR instance.

## Parameters

in	<i>instance</i>	- LPTMR instance number
out	<i>compareValueByCount</i>	- Pointer to current compare value, in counter ticks

16.82.4.5 void `LPTMR_DRV_GetCompareValueByUs ( const uint32_t instance, uint32_t *const compareValueUs )`

Get the compare value in microseconds, of a LPTMR instance.

## Parameters

in	<i>instance</i>	- LPTMR instance number
out	<i>compareValue</i> ↔ <i>Us</i>	- Pointer to current compare value, in microseconds

#### 16.82.4.6 void LPTMR\_DRV\_GetConfig ( const uint32\_t *instance*, lptmr\_config\_t \*const *config* )

Get the current configuration of a LPTMR instance.

##### Parameters

in	<i>instance</i>	- LPTMR instance number
out	<i>config</i>	- Pointer to the output configuration structure

#### 16.82.4.7 uint16\_t LPTMR\_DRV\_GetCounterValueByCount ( const uint32\_t *instance* )

Get the current counter value in counter tick units.

##### Parameters

in	<i>instance</i>	- LPTMR instance number
----	-----------------	-------------------------

##### Returns

The current counter value

#### 16.82.4.8 void LPTMR\_DRV\_Init ( const uint32\_t *instance*, const lptmr\_config\_t \*const *config*, const bool *startCounter* )

Initialize a LPTMR instance with values from an input configuration structure.

When (counterUnits == LPTMR\_COUNTER\_UNITS\_MICROSECONDS) the function will automatically configure the timer for the input compareValue in microseconds. The input parameters for 'prescaler' and 'bypassPrescaler' will be ignored - their values will be adapted by the function, to best fit the input compareValue (in microseconds) for the operating clock frequency.

LPTMR\_COUNTER\_UNITS\_MICROSECONDS may only be used for LPTMR\_WORKMODE\_TIMER mode. Otherwise the function shall not convert 'compareValue' in ticks and this is likely to cause erroneous behavior.

When (counterUnits == LPTMR\_COUNTER\_UNITS\_TICKS) the function will use the 'prescaler' and 'bypassPrescaler' provided in the input configuration structure.

##### Parameters

in	<i>instance</i>	- LPTMR instance number
in	<i>config</i>	- Pointer to the input configuration structure
in	<i>startCounter</i>	- Flag for starting the counter immediately after configuration

#### 16.82.4.9 void LPTMR\_DRV\_InitConfigStruct ( lptmr\_config\_t \*const *config* )

Initialize a configuration structure with default values.

##### Parameters

out	<i>config</i>	- Pointer to the configuration structure to be initialized
-----	---------------	--

#### 16.82.4.10 bool LPTMR\_DRV\_IsRunning ( const uint32\_t *instance* )

Get the run state of a LPTMR instance.

## Parameters

in	<i>instance</i>	- LPTMR instance number
----	-----------------	-------------------------

## Returns

The run state of the LPTMR instance:

- true: Timer/Counter started
- false: Timer/Counter stopped

#### 16.82.4.11 `status_t LPTMR_DRV_SetCompareValueByCount ( const uint32_t instance, const uint16_t compareValueByCount )`

Set the compare value in counter tick units, for a LPTMR instance.

## Parameters

in	<i>instance</i>	- LPTMR instance number
in	<i>compareValue↔ ByCount</i>	- The compare value in counter ticks, to be written

## Returns

Operation status:

- STATUS\_SUCCESS: completed successfully
- STATUS\_ERROR: cannot reconfigure compare value (TCF not set)
- STATUS\_TIMEOUT: compare value greater then current counter value

#### 16.82.4.12 `status_t LPTMR_DRV_SetCompareValueByUs ( const uint32_t instance, const uint32_t compareValueUs )`

Set the compare value for Timer Mode in microseconds, for a LPTMR instance.

## Parameters

in	<i>instance</i>	- LPTMR peripheral instance number
in	<i>compareValue↔ Us</i>	- Compare value in microseconds

## Returns

Operation status:

- STATUS\_SUCCESS: completed successfully
- STATUS\_ERROR: cannot reconfigure compare value
- STATUS\_TIMEOUT: compare value greater then current counter value

#### 16.82.4.13 `void LPTMR_DRV_SetConfig ( const uint32_t instance, const lptmr_config_t *const config )`

Configure a LPTMR instance.

When (counterUnits == LPTMR\_COUNTER\_UNITS\_MICROSECONDS) the function will automatically configure the timer for the input compareValue in microseconds. The input parameters for 'prescaler' and 'bypassPrescaler' will be ignored - their values will be adapted by the function, to best fit the input compareValue (in microseconds) for the operating clock frequency.

LPTMR\_COUNTER\_UNITS\_MICROSECONDS may only be used for LPTMR\_WORKMODE\_TIMER mode. Otherwise the function shall not convert 'compareValue' in ticks and this is likely to cause erroneous behavior.

When (counterUnits == LPTMR\_COUNTER\_UNITS\_TICKS) the function will use the 'prescaler' and 'bypass↔Prescaler' provided in the input configuration structure.

**Parameters**

in	<i>instance</i>	- LPTMR instance number
in	<i>config</i>	- Pointer to the input configuration structure

**16.82.4.14** void LPTMR\_DRV\_SetInterrupt ( const uint32\_t *instance*, const bool *enableInterrupt* )

Enable/disable the LPTMR interrupt.

**Parameters**

in	<i>instance</i>	- LPTMR instance number
in	<i>enableInterrupt</i>	- The new state of the LPTMR interrupt enable flag.

**16.82.4.15** void LPTMR\_DRV\_SetPinConfiguration ( const uint32\_t *instance*, const lptmr\_pinselect\_t *pinSelect*, const lptmr\_pinpolarity\_t *pinPolarity* )

Set the Input Pin configuration for Pulse Counter mode.

**Parameters**

in	<i>instance</i>	- LPTMR instance number
in	<i>pinSelect</i>	- LPTMR pin selection
in	<i>pinPolarity</i>	- Polarity on which to increment counter (rising/falling edge)

**16.82.4.16** void LPTMR\_DRV\_StartCounter ( const uint32\_t *instance* )

Enable the LPTMR / Start the counter.

**Parameters**

in	<i>instance</i>	- LPTMR instance number
----	-----------------	-------------------------

**16.82.4.17** void LPTMR\_DRV\_StopCounter ( const uint32\_t *instance* )

Disable the LPTMR / Stop the counter.

**Parameters**

in	<i>instance</i>	- LPTMR instance number
----	-----------------	-------------------------

## 16.83 LPUART Driver

### 16.83.1 Detailed Description

This module covers the functionality of the Low Power Universal Asynchronous Receiver-Transmitter (LPUART) peripheral driver.

The LPUART driver implements serial communication using the LPUART module in the S32K1xx platforms.

#### Features

- Interrupt based, DMA based and polling communication
- Provides blocking and non-blocking transmit and receive functions
- Configurable baud rate
- 8/9/10 bits per char

#### Functionality

In order to use the LPUART driver it must be first initialized, using [LPUART\\_DRV\\_Init\(\)](#) function. Once initialized, it cannot be initialized again for the same LPUART module instance until it is de-initialized, using [LPUART\\_DRV\\_Deinit\(\)](#). The initialization function does the following operations:

- sets the baud rate
- sets parity/bit count/stop bits count
- initializes the state structure for the current instance Different LPUART module instances can function independently of each other.

#### Interrupt-based communication

After initialization, a serial communication can be triggered by calling [LPUART\\_DRV\\_SendData\(\)](#) function; this will save the reference of the data buffer received as parameter in the internal tx buffer pointer, then copy the first byte to the data register. The transmitter then automatically shifts the data and triggers a 'Transmit buffer empty' interrupt when all bits are shifted. The drivers interrupt handler takes care of transmitting the next byte in the buffer, by increasing the data pointer and decreasing the data size. The same sequence of operations is executed until all bytes in the tx buffer have been transmitted.

Similarly, data reception is triggered by calling [LPUART\\_DRV\\_ReceiveData\(\)](#) function, passing the rx buffer as parameter. When the receiver copies the received bits in the data register, the 'Receive buffer full' interrupt is triggered; the driver irq handler clears the flag by reading the received byte, saves it in the rx buffer, then increments the data pointer and decrements the data size. This is repeated until all bytes are received.

The workflow applies to send/receive operations using blocking method (triggered by [LPUART\\_DRV\\_SendDataBlocking\(\)](#) and [LPUART\\_DRV\\_ReceiveDataBlocking\(\)](#)), with the single difference that the send/receive function will not return until the send/receive operation is complete (all bytes are successfully transferred or a timeout occurred). The timeout for the blocking method is passed as parameter by the user.

#### DMA-based communication

In DMA operation, both blocking and non-blocking transmission methods configure a DMA channel to copy data from the buffer to the data register (for tx), or viceversa (for rx). The driver assumes the DMA channel is already allocated and the proper requests are routed to it via DMAMUX. After configuring the DMA channel, the driver enables DMA requests for rx/tx, then the DMA engine takes care of moving data to/from the data buffer.

#### Polling mode

The driver also provides polling methods for send and receive ([LPUART\\_DRV\\_SendDataPolling\(\)](#) and [LPUART\\_DRV\\_ReceiveDataPolling\(\)](#)). These functions are blocking (return only when the transfer is complete) and do not use interrupt or DMA services. The tx buffer empty and rx buffer full flags are polled by software in order to copy data to/from the data register.

## Error handling

The driver treats the following errors on reception:

- rx overrun
- parity error
- framing error
- noise error

In case any of these error events occur on the rx line during an ongoing reception, the transfer is aborted and rx status is updated accordingly. [LPUART\\_DRV\\_GetReceiveStatus\(\)](#) function can be called to retrieve the status of the last reception. If a receive callback is installed, it is called right after aborting the current transfer (with `UART_EVENT_ERROR` parameter).

## Callbacks

The driver provides callback notifications for asynchronous transfers. [LPUART\\_DRV\\_InstallTxCallback\(\)](#) function can be used for installing a callback routine to be called when the transmission is finished. The tx callback is called twice: first when the tx buffer becomes empty (no more data to be transmitted) - at this point the application can call [LPUART\\_DRV\\_SetTxBuffer\(\)](#) inside the callback in order to provide more data, resulting in a continuous transmission; if there is no more data to be transmitted, the callback is called again when the transmission is complete (all the bytes have been shifted out on the line). The event parameter in the callback signature differentiates these two calls - the values are `UART_EVENT_TX_EMPTY` and `UART_EVENT_END_TRANSFER`, respectively.

Similarly, [LPUART\\_DRV\\_InstallRxCallback\(\)](#) installs a callback routine for reception. This callback is called twice (`UART_EVENT_RX_FULL` and `UART_EVENT_END_TRANSFER`); if a new buffer is provided within the first callback call ([LPUART\\_DRV\\_SetRxBuffer\(\)](#)), the reception will continue without interruption. In case of an error detected during an ongoing reception, the transfer is aborted and the callback is called with `UART_EVENT_ERROR` parameter. The driver treats rx overrun, parity, framing and noise errors.

## Important Notes

- Before using the LPUART driver the module clock must be configured
- The driver enables the interrupts for the corresponding LPUART module, but any interrupt priority must be done by the application
- The board specific configurations must be done prior to driver calls; the driver has no influence on the functionality of the rx/tx pins - they must be configured by application
- DMA module has to be initialized prior to LPUART usage in DMA mode; also, DMA channels need to be allocated for LPUART usage by the application (the driver only takes care of configuring the DMA channels received in the configuration structure)
- For 9/10 bits characters, the application must provide the appropriate buffers; the size of the tx/rx buffers in this scenario needs to be an even number, as the transferred characters will be split in two bytes (bit 8 for 9-bits chars and bits 8 & 9 for 10-bits chars will be stored in the subsequent byte). 9/10 bits chars are only supported in interrupt-based and polling communications
- For 10-bits word length, parity cannot be enabled.
- When the vector table is not in ram (**flash\_vector\_table** = 1):
  - `INT_SYS_InstallHandler` shall check if the function pointer provided as parameter for the new handler is already present in the vector table for the given IRQ number.
  - The user will be required to manually add the correct handlers in the startup files



### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\drivers\src\lpuart\lpuart_driver.c
${S32SDK_PATH}\platform\drivers\src\lpuart\lpuart_hw_access.c
${S32SDK_PATH}\platform\drivers\src\lpuart\lpuart_irq.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc\
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

[Clock Manager Interrupt Manager \(Interrupt\) Enhanced Direct Memory Access \(eDMA\)](#)

#### Data Structures

- struct [lpuart\\_state\\_t](#)  
*Runtime state of the LPUART driver. [More...](#)*
- struct [lpuart\\_user\\_config\\_t](#)  
*LPUART configuration structure. [More...](#)*

#### Enumerations

- enum [lpuart\\_transfer\\_type\\_t](#) { LPUART\_USING\_DMA = 0, LPUART\_USING\_INTERRUPTS }  
*Type of LPUART transfer (based on interrupts or DMA).*
- enum [lpuart\\_bit\\_count\\_per\\_char\\_t](#) { LPUART\_8\_BITS\_PER\_CHAR = 0x0U, LPUART\_9\_BITS\_PER\_CHAR = 0x1U, LPUART\_10\_BITS\_PER\_CHAR = 0x2U }  
*LPUART number of bits in a character.*
- enum [lpuart\\_parity\\_mode\\_t](#) { LPUART\_PARITY\_DISABLED = 0x0U, LPUART\_PARITY\_EVEN = 0x2U, LPUART\_PARITY\_ODD = 0x3U }  
*LPUART parity mode.*
- enum [lpuart\\_stop\\_bit\\_count\\_t](#) { LPUART\_ONE\_STOP\_BIT = 0x0U, LPUART\_TWO\_STOP\_BIT = 0x1U }  
*LPUART number of stop bits.*

#### LPUART Driver

- void [LPUART\\_DRV\\_GetDefaultConfig](#) ([lpuart\\_user\\_config\\_t](#) \*lpuartUserConfig)  
*Initializes the LPUART configuration structure with default values.*
- status\_t [LPUART\\_DRV\\_Init](#) (uint32\_t instance, [lpuart\\_state\\_t](#) \*lpuartStatePtr, const [lpuart\\_user\\_config\\_t](#) \*lpuartUserConfig)  
*Initializes an LPUART operation instance.*
- status\_t [LPUART\\_DRV\\_Deinit](#) (uint32\_t instance)  
*Shuts down the LPUART by disabling interrupts and transmitter/receiver.*
- uart\_callback\_t [LPUART\\_DRV\\_InstallRxCallback](#) (uint32\_t instance, uart\_callback\_t function, void \*callbackParam)

*Installs callback function for the LPUART receive.*

- `uart_callback_t LPUART_DRV_InstallTxCallback` (`uint32_t` instance, `uart_callback_t` function, `void *callbackParam`)

*Installs callback function for the LPUART transmit.*

- `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.*

- `status_t LPUART_DRV_SendDataPolling` (`uint32_t` instance, `const uint8_t *txBuff`, `uint32_t txSize`)

*Send out multiple bytes of data using polling method.*

- `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. 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.*

- `status_t LPUART_DRV_GetTransmitStatus` (`uint32_t` instance, `uint32_t *bytesRemaining`)

*Returns whether the previous transmit is complete.*

- `status_t LPUART_DRV_AbortSendingData` (`uint32_t` instance)

*Terminates a non-blocking transmission early.*

- `status_t LPUART_DRV_ReceiveDataBlocking` (`uint32_t` instance, `uint8_t *rxBuff`, `uint32_t rxSize`, `uint32_t timeout`)

*Gets data from the LPUART module by using a blocking method. Blocking means that the function does not return until the receive is complete.*

- `status_t LPUART_DRV_ReceiveDataPolling` (`uint32_t` instance, `uint8_t *rxBuff`, `uint32_t rxSize`)

*Receive multiple bytes of data using polling method.*

- `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. 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.*

- `status_t LPUART_DRV_GetReceiveStatus` (`uint32_t` instance, `uint32_t *bytesRemaining`)

*Returns whether the previous receive is complete.*

- `status_t LPUART_DRV_AbortReceivingData` (`uint32_t` instance)

*Terminates a non-blocking receive early.*

- `status_t LPUART_DRV_SetBaudRate` (`uint32_t` instance, `uint32_t desiredBaudRate`)

*Configures the LPUART baud rate.*

- `void LPUART_DRV_GetBaudRate` (`uint32_t` instance, `uint32_t *configuredBaudRate`)

*Returns the LPUART baud rate.*

- `status_t LPUART_DRV_SetTxBuffer` (`uint32_t` instance, `const uint8_t *txBuff`, `uint32_t txSize`)

*Sets the internal driver reference to the tx buffer.*

- `status_t LPUART_DRV_SetRxBuffer` (`uint32_t` instance, `uint8_t *rxBuff`, `uint32_t rxSize`)

*Sets the internal driver reference to the rx buffer.*

## 16.83.2 Data Structure Documentation

### 16.83.2.1 struct `lpuart_state_t`

Runtime state of the LPUART driver.

Note that the caller provides memory for the driver state structures during initialization because the driver does not statically allocate memory.

Implements : `lpuart_state_t` Class

Definition at line 89 of file `lpuart_driver.h`.

## Data Fields

- `const uint8_t * txBuff`
- `uint8_t * rxBuff`
- `volatile uint32_t txSize`
- `volatile uint32_t rxSize`
- `volatile bool isTxBusy`
- `volatile bool isRxBusy`
- `volatile bool isTxBlocking`
- `volatile bool isRxBlocking`
- `lpuart_bit_count_per_char_t bitCountPerChar`
- `uart_callback_t rxCallback`
- `void * rxCallbackParam`
- `uart_callback_t txCallback`
- `void * txCallbackParam`
- `lpuart_transfer_type_t transferType`
- `semaphore_t rxComplete`
- `semaphore_t txComplete`
- `volatile status_t transmitStatus`
- `volatile status_t receiveStatus`

## Field Documentation

16.83.2.1.1 `lpuart_bit_count_per_char_t bitCountPerChar`

number of bits in a char (8/9/10)

Definition at line 99 of file `lpuart_driver.h`.

16.83.2.1.2 `volatile bool isRxBlocking`

True if receive is blocking transaction.

Definition at line 98 of file `lpuart_driver.h`.

16.83.2.1.3 `volatile bool isRxBusy`

True if there is an active receive.

Definition at line 96 of file `lpuart_driver.h`.

16.83.2.1.4 `volatile bool isTxBlocking`

True if transmit is blocking transaction.

Definition at line 97 of file `lpuart_driver.h`.

16.83.2.1.5 `volatile bool isTxBusy`

True if there is an active transmit.

Definition at line 95 of file `lpuart_driver.h`.

16.83.2.1.6 `volatile status_t receiveStatus`

Status of last driver receive operation

Definition at line 120 of file `lpuart_driver.h`.

16.83.2.1.7 `uint8_t* rxBuff`

The buffer of received data.

Definition at line 92 of file `lpuart_driver.h`.

**16.83.2.1.8    `uart_callback_t rxCallback`**

Callback to invoke for data receive Note: when the transmission is interrupt based, the callback is being called upon receiving a byte; when DMA transmission is used, the bytes are copied to the rx buffer by the DMA engine and the callback is called when all the bytes have been transferred.

Definition at line 100 of file `lpuart_driver.h`.

**16.83.2.1.9    `void* rxCallbackParam`**

Receive callback parameter pointer.

Definition at line 105 of file `lpuart_driver.h`.

**16.83.2.1.10   `semaphore_t rxComplete`**

Synchronization object for blocking Rx timeout condition

Definition at line 117 of file `lpuart_driver.h`.

**16.83.2.1.11   `volatile uint32_t rxSize`**

The remaining number of bytes to be received.

Definition at line 94 of file `lpuart_driver.h`.

**16.83.2.1.12   `lpuart_transfer_type_t transferType`**

Type of LPUART transfer (interrupt/dma based)

Definition at line 112 of file `lpuart_driver.h`.

**16.83.2.1.13   `volatile status_t transmitStatus`**

Status of last driver transmit operation

Definition at line 119 of file `lpuart_driver.h`.

**16.83.2.1.14   `const uint8_t* txBuff`**

The buffer of data being sent.

Definition at line 91 of file `lpuart_driver.h`.

**16.83.2.1.15   `uart_callback_t txCallback`**

Callback to invoke for data send Note: when the transmission is interrupt based, the callback is being called upon sending a byte; when DMA transmission is used, the bytes are copied to the tx buffer by the DMA engine and the callback is called when all the bytes have been transferred.

Definition at line 106 of file `lpuart_driver.h`.

**16.83.2.1.16   `void* txCallbackParam`**

Transmit callback parameter pointer.

Definition at line 111 of file `lpuart_driver.h`.

**16.83.2.1.17   `semaphore_t txComplete`**

Synchronization object for blocking Tx timeout condition

Definition at line 118 of file `lpuart_driver.h`.

**16.83.2.1.18 volatile uint32\_t txSize**

The remaining number of bytes to be transmitted.

Definition at line 93 of file lpuart\_driver.h.

**16.83.2.2 struct lpuart\_user\_config\_t**

LPUART configuration structure.

Implements : lpuart\_user\_config\_t\_Class

Definition at line 127 of file lpuart\_driver.h.

**Data Fields**

- uint32\_t [baudRate](#)
- [lpuart\\_parity\\_mode\\_t](#) parityMode
- [lpuart\\_stop\\_bit\\_count\\_t](#) stopBitCount
- [lpuart\\_bit\\_count\\_per\\_char\\_t](#) bitCountPerChar
- [lpuart\\_transfer\\_type\\_t](#) transferType
- uint8\_t rxDMAChannel
- uint8\_t txDMAChannel

**Field Documentation****16.83.2.2.1 uint32\_t baudRate**

LPUART baud rate

Definition at line 129 of file lpuart\_driver.h.

**16.83.2.2.2 lpuart\_bit\_count\_per\_char\_t bitCountPerChar**

number of bits in a character (8-default, 9 or 10); for 9/10 bits chars, users must provide appropriate buffers to the send/receive functions (bits 8/9 in subsequent bytes); for DMA transmission only 8-bit char is supported.

Definition at line 132 of file lpuart\_driver.h.

**16.83.2.2.3 lpuart\_parity\_mode\_t parityMode**

parity mode, disabled (default), even, odd

Definition at line 130 of file lpuart\_driver.h.

**16.83.2.2.4 uint8\_t rxDMAChannel**

Channel number for DMA rx channel. If DMA mode isn't used this field will be ignored.

Definition at line 137 of file lpuart\_driver.h.

**16.83.2.2.5 lpuart\_stop\_bit\_count\_t stopBitCount**

number of stop bits, 1 stop bit (default) or 2 stop bits

Definition at line 131 of file lpuart\_driver.h.

**16.83.2.2.6 lpuart\_transfer\_type\_t transferType**

Type of LPUART transfer (interrupt/dma based)

Definition at line 136 of file lpuart\_driver.h.

**16.83.2.2.7 uint8\_t txDMAChannel**

Channel number for DMA tx channel. If DMA mode isn't used this field will be ignored.

Definition at line 139 of file lpuart\_driver.h.

### 16.83.3 Enumeration Type Documentation

#### 16.83.3.1 enum lpuart\_bit\_count\_per\_char\_t

LPUART number of bits in a character.

Implements : lpuart\_bit\_count\_per\_char\_t\_Class

#### Enumerator

**LPUART\_8\_BITS\_PER\_CHAR** 8-bit data characters  
**LPUART\_9\_BITS\_PER\_CHAR** 9-bit data characters  
**LPUART\_10\_BITS\_PER\_CHAR** 10-bit data characters

Definition at line 53 of file lpuart\_driver.h.

#### 16.83.3.2 enum lpuart\_parity\_mode\_t

LPUART parity mode.

Implements : lpuart\_parity\_mode\_t\_Class

#### Enumerator

**LPUART\_PARITY\_DISABLED** parity disabled  
**LPUART\_PARITY\_EVEN** parity enabled, type even, bit setting: PE|PT = 10  
**LPUART\_PARITY\_ODD** parity enabled, type odd, bit setting: PE|PT = 11

Definition at line 64 of file lpuart\_driver.h.

#### 16.83.3.3 enum lpuart\_stop\_bit\_count\_t

LPUART number of stop bits.

Implements : lpuart\_stop\_bit\_count\_t\_Class

#### Enumerator

**LPUART\_ONE\_STOP\_BIT** one stop bit  
**LPUART\_TWO\_STOP\_BIT** two stop bits

Definition at line 75 of file lpuart\_driver.h.

#### 16.83.3.4 enum lpuart\_transfer\_type\_t

Type of LPUART transfer (based on interrupts or DMA).

Implements : lpuart\_transfer\_type\_t\_Class

#### Enumerator

**LPUART\_USING\_DMA** The driver will use DMA to perform UART transfer  
**LPUART\_USING\_INTERRUPTS** The driver will use interrupts to perform UART transfer

Definition at line 43 of file lpuart\_driver.h.

### 16.83.4 Function Documentation

#### 16.83.4.1 status\_t LPUART\_DRV\_AbortReceivingData ( uint32\_t instance )

Terminates a non-blocking receive early.

## Parameters

<i>instance</i>	LPUART instance number
-----------------	------------------------

## Returns

Whether the receiving was successful or not.

Definition at line 891 of file lpuart\_driver.c.

#### 16.83.4.2 status\_t LPUART\_DRV\_AbortSendingData ( uint32\_t *instance* )

Terminates a non-blocking transmission early.

## Parameters

<i>instance</i>	LPUART instance number
-----------------	------------------------

## Returns

Whether the aborting is successful or not.

Definition at line 580 of file lpuart\_driver.c.

#### 16.83.4.3 status\_t LPUART\_DRV\_Deinit ( uint32\_t *instance* )

Shuts down the LPUART by disabling interrupts and transmitter/receiver.

## Parameters

<i>instance</i>	LPUART instance number
-----------------	------------------------

## Returns

STATUS\_SUCCESS if successful; STATUS\_ERROR if an error occurred

Definition at line 278 of file lpuart\_driver.c.

#### 16.83.4.4 void LPUART\_DRV\_GetBaudRate ( uint32\_t *instance*, uint32\_t \* *configuredBaudRate* )

Returns the LPUART baud rate.

This function returns the LPUART configured baud rate.

## Parameters

	<i>instance</i>	LPUART instance number.
out	<i>configuredBaudRate</i>	LPUART configured baud rate.

Definition at line 1036 of file lpuart\_driver.c.

#### 16.83.4.5 void LPUART\_DRV\_GetDefaultConfig ( lpuart\_user\_config\_t \* *lpuartUserConfig* )

Initializes the LPUART configuration structure with default values.

This function initializes a configuration structure received from the application with default values.

## Parameters

<i>lpuartUserConfig</i>	user configuration structure of type <a href="#">lpuart_user_config_t</a>
-------------------------	---

Definition at line 145 of file `lpuart_driver.c`.

16.83.4.6 `status_t LPUART_DRV_GetReceiveStatus ( uint32_t instance, uint32_t * bytesRemaining )`

Returns whether the previous receive is complete.

#### Parameters

<i>instance</i>	LPUART instance number
<i>bytesRemaining</i>	pointer to value that is filled with the number of bytes that still need to be received in the active transfer.

#### Note

In DMA mode, this parameter may not be accurate, in case the transfer completes right after calling this function; in this edge-case, the parameter will reflect the initial transfer size, due to automatic reloading of the major loop count in the DMA transfer descriptor.

#### Returns

The receive status.

#### Return values

<code>STATUS_SUCCESS</code>	the receive has completed successfully.
<code>STATUS_BUSY</code>	the receive is still in progress. <i>bytesReceived</i> will be filled with the number of bytes that have been received so far.
<code>STATUS_UART_ABORTED</code>	The receive was aborted.
<code>STATUS_TIMEOUT</code>	A timeout was reached.
<code>STATUS_UART_RX_OVERRUN, STATUS_UART_FRAMING_ERROR, STATUS_UART_PARITY_ERROR, or</code>	<code>STATUS_UART_NOISE_ERROR, STATUS_ERROR</code> An error occurred during reception.

Definition at line 846 of file `lpuart_driver.c`.

16.83.4.7 `status_t LPUART_DRV_GetTransmitStatus ( uint32_t instance, uint32_t * bytesRemaining )`

Returns whether the previous transmit is complete.

#### Parameters

<i>instance</i>	LPUART instance number
<i>bytesRemaining</i>	Pointer to value that is populated with the number of bytes that have been sent in the active transfer

#### Note

In DMA mode, this parameter may not be accurate, in case the transfer completes right after calling this function; in this edge-case, the parameter will reflect the initial transfer size, due to automatic reloading of the major loop count in the DMA transfer descriptor.

#### Returns

The transmit status.



## Return values

<i>STATUS_SUCCESS</i>	The transmit has completed successfully.
<i>STATUS_BUSY</i>	The transmit is still in progress. <i>bytesRemaining</i> will be filled with the number of bytes that are yet to be transmitted.
<i>STATUS_UART_ABORTED</i>	The transmit was aborted.
<i>STATUS_TIMEOUT</i>	A timeout was reached.
<i>STATUS_ERROR</i>	An error occurred.

Definition at line 534 of file `lpuart_driver.c`.

**16.83.4.8** `status_t LPUART_DRV_Init ( uint32_t instance, lpuart_state_t * lpuartStatePtr, const lpuart_user_config_t * lpuartUserConfig )`

Initializes an LPUART operation instance.

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 <code>lpuart_user_config_t</code>
<i>lpuartStatePtr</i>	pointer to the LPUART driver state structure

## Returns

`STATUS_SUCCESS` if successful; `STATUS_ERROR` if an error occurred

Definition at line 181 of file `lpuart_driver.c`.

**16.83.4.9** `uart_callback_t LPUART_DRV_InstallRxCallback ( uint32_t instance, uart_callback_t function, void * callbackParam )`

Installs callback function for the LPUART receive.

## 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.

## Returns

Former LPUART receive callback function pointer.

Definition at line 319 of file `lpuart_driver.c`.

**16.83.4.10** `uart_callback_t LPUART_DRV_InstallTxCallback ( uint32_t instance, uart_callback_t function, void * callbackParam )`

Installs callback function for the LPUART transmit.

**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.

Definition at line 342 of file lpuart\_driver.c.

#### 16.83.4.11 `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. 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**

STATUS\_SUCCESS if successful; STATUS\_BUSY if the resource is busy

Definition at line 803 of file lpuart\_driver.c.

#### 16.83.4.12 `status_t LPUART_DRV_ReceiveDataBlocking ( uint32_t instance, uint8_t * rxBuff, uint32_t rxSize, uint32_t timeout )`

Gets data from the LPUART module by using a blocking method. 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 in milliseconds

**Returns**

STATUS\_SUCCESS if successful; STATUS\_TIMEOUT if the timeout was reached; STATUS\_BUSY if the resource is busy; STATUS\_UART\_FRAMING\_ERROR if a framing error occurred; STATUS\_UART\_NOISE\_ERROR if a noise error occurred; STATUS\_UART\_PARITY\_ERROR if a parity error occurred; STATUS\_UART\_RX\_OVERRUN if an overrun error occurred; STATUS\_ERROR if a DMA error occurred;

Definition at line 618 of file lpuart\_driver.c.

#### 16.83.4.13 `status_t LPUART_DRV_ReceiveDataPolling ( uint32_t instance, uint8_t * rxBuff, uint32_t rxSize )`

Receive multiple bytes of data using polling method.

## Parameters

<i>instance</i>	LPUART instance number.
<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

STATUS\_SUCCESS if the transaction is successful; STATUS\_BUSY if the resource is busy; STATUS\_UART\_RX\_OVERRUN if an overrun error occurred.

Definition at line 683 of file lpuart\_driver.c.

#### 16.83.4.14 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. 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

STATUS\_SUCCESS if successful; STATUS\_BUSY if the resource is busy;

Definition at line 490 of file lpuart\_driver.c.

#### 16.83.4.15 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.

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 in milliseconds

## Returns

STATUS\_SUCCESS if successful; STATUS\_TIMEOUT if the timeout was reached; STATUS\_BUSY if the resource is busy; STATUS\_ERROR if an error occurred

Definition at line 365 of file lpuart\_driver.c.

#### 16.83.4.16 status\_t LPUART\_DRV\_SendDataPolling ( uint32\_t instance, const uint8\_t \* txBuff, uint32\_t txSize )

Send out multiple bytes of data using polling method.

**Parameters**

<i>instance</i>	LPUART instance number.
<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**

STATUS\_SUCCESS if successful; STATUS\_BUSY if the resource is busy;

Definition at line 430 of file lpuart\_driver.c.

**16.83.4.17** `status_t LPUART_DRV_SetBaudRate ( uint32_t instance, uint32_t desiredBaudRate )`

Configures the LPUART baud rate.

This function configures the LPUART baud rate. 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>instance</i>	LPUART instance number.
<i>desiredBaudRate</i>	LPUART desired baud rate.

**Returns**

STATUS\_BUSY if called during an on-going transfer, STATUS\_SUCCESS otherwise

Definition at line 931 of file lpuart\_driver.c.

**16.83.4.18** `status_t LPUART_DRV_SetRxBuffer ( uint32_t instance, uint8_t * rxBuff, uint32_t rxSize )`

Sets the internal driver reference to the rx buffer.

This function can be called from the rx callback to provide the driver with a new buffer, for continuous reception.

**Parameters**

<i>instance</i>	LPUART instance number
<i>rxBuff</i>	destination buffer containing 8-bit data chars to receive
<i>rxSize</i>	the number of bytes to receive

**Returns**

STATUS\_SUCCESS

Definition at line 1089 of file lpuart\_driver.c.

**16.83.4.19** `status_t LPUART_DRV_SetTxBuffer ( uint32_t instance, const uint8_t * txBuff, uint32_t txSize )`

Sets the internal driver reference to the tx buffer.

This function can be called from the tx callback to provide the driver with a new buffer, for continuous transmission.

**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**

STATUS\_SUCCESS

Definition at line 1065 of file lpuart\_driver.c.

## 16.84 Local Interconnect Network (LIN)

### 16.84.1 Detailed Description

The S32 SDK provides both driver and middleware layers for the Local Interconnect Network (LIN) protocol, emulated on top of LPUART serial communication IP.

#### Modules

- [LIN Driver](#)

*This section describes the programming interface of the Peripheral driver for LIN.*

- [LIN Stack](#)

*This section covers the functionality of the LIN Stack middleware layer in S32 SDK.*

## 16.85 Low Power Inter-Integrated Circuit (LPI2C)

### 16.85.1 Detailed Description

The LPI2C is a low power Inter-Integrated Circuit (I2C) module that supports an efficient interface to an I2C bus as a master and/or a slave.

#### Modules

- [LPI2C Driver](#)

*Low Power Inter-Integrated Circuit (LPI2C) Peripheral Driver.*

## 16.86 Low Power Interrupt Timer (LPIT)

### 16.86.1 Detailed Description

The Low Power Periodic Interrupt Timer (LPIT) is a multi-channel timer module generating independent pre-trigger and trigger outputs. These timer channels can operate individually or can be chained together. The LPIT can operate in low power modes if configured to do so. The pre-trigger and trigger outputs can be used to trigger other modules on the device.

The S32 SDK provides Peripheral Drivers for the Low Power Interrupt Timer (LPIT) module of S32 devices.

#### Modules

- [LPIT Driver](#)

*Low Power Interrupt Timer Peripheral Driver.*



## 16.87 Low Power Serial Peripheral Interface (LPSPi)

### 16.87.1 Detailed Description

Low Power Serial Peripheral Interface (LPSPi) Peripheral Driver.

The LPSPi driver allows communication on an SPI bus using the LPSPi module in the S32K1xx processors.

#### Features

- Interrupt based
- Master or slave operation
- Provides blocking and non-blocking transmit and receive functions
- RX and TX hardware buffers (4 words)
- 4 configurable chip select
- Configurable baud rate

#### How to integrate LPSPi in your application

In order to use the LPSPi driver it must be first initialized in either master or slave mode, using functions [LPSPi\\_DRV\\_MasterInit\(\)](#) or [LPSPi\\_DRV\\_SlaveInit\(\)](#). Once initialized, it cannot be initialized again for the same LPSPi module instance until it is de-initialized, using [LPSPi\\_DRV\\_MasterDeinit\(\)](#) or [LPSPi\\_DRV\\_SlaveDeinit\(\)](#). Different LPSPi module instances can function independently of each other.

In each mode (master/slave) are available two types of transfers: blocking and non-blocking. The functions which initiate blocking transfers will configure the time out for transmission. If time expires [LPSPi\\_MasterTransferBlocking\(\)](#) or [LPSPi\\_SlaveTransferBlocking\(\)](#) will return error and the transmission will be aborted.

Depending on frame size receive and transmit buffers must be aligned as is presented in the next table:

Bits/frame	less or equal with 8	between 9 and 16	more than 16
Alignment	1 byte	2 bytes	4 bytes

This alignment requirements should be taken into consideration when "transferByteCount" is configured. For a better understanding these are some examples of how to calculate the right value to "transferByteCount":

Bits/frame	number of frames	bytes per frame	transferByteCount
8	10	1	10
10	10	2	20
24	10	4	40
32	10	4	40
40	10	8	80
64	10	8	80

If frame size is bigger than 32 bits MSB/LSB option is applicable for each uint32\_t, not for the entire frame. The application should ensure the uint32\_t order in buffers, depending on MSB/LSB configuration.

#### ## Important Notes ##

- The driver enables the interrupts for the corresponding LPSPi module, but any interrupt priority setting must be done by the application.
- The watermarks will be set by the application.
- The driver will configure SCK to PCS delay, PCS to SCK delay, delay between transfers with default values. If your application needs other values for this parameters [LPSPi\\_DRV\\_MasterSetDelay](#) function can be used.
- The driver cannot be used in the case SPI transfer in slave mode over DMA with invalid address of tx buffer, the driver will never finish the transfer.

- The driver cannot be used with a configuration with bit/frame greater than 32 bits and MSB endianness in either slave or master mode.
- LPSPI2 instance is only available on 64-pin variant of S32K14xW and not available on 48-pin variant. If you need a frame larger than 32 bits with MSB the application must handle the data positioning.

### Example code

```
const lpspi_master_config_t Send_MasterConfig0 = {
    .bitsPerSec = 50000U,
    .whichPcs = LPSPI_PCS0,
    .pcsPolarity = LPSPI_ACTIVE_HIGH,
    .isPcsContinuous = false,
    .bitcount = 8U,
    .lpspiSrcClk = 8000000U,
    .clkPhase = LPSPI_CLOCK_PHASE_1ST_EDGE,
    .clkPolarity = LPSPI_SCK_ACTIVE_HIGH,
    .lsbFirst = false,
    .transferType = LPSPI_USING_INTERRUPTS,
};

const lpspi_slave_config_t Receive_SlaveConfig0 = {
    .pcsPolarity = LPSPI_ACTIVE_HIGH,
    .bitcount = 8U,
    .clkPhase = LPSPI_CLOCK_PHASE_1ST_EDGE,
    .whichPcs = LPSPI_PCS0,
    .clkPolarity = LPSPI_SCK_ACTIVE_HIGH,
    .lsbFirst = false,
    .transferType = LPSPI_USING_INTERRUPTS,
};

/* Initialize clock and pins */
LPSPI_DRV_MasterInit(0U, &masterState, &Send_MasterConfig0);
/* Set delay between transfer, PCStoSCK and SCKtoPCS to 10 microseconds. */
LPSPI_DRV_MasterSetDelay(0U, 10U, 10U, 10U);
/* Initialize LPSPI1 (Slave)*/
LPSPI_DRV_SlaveInit(1U, &slaveState, &Receive_SlaveConfig0);
/* Allocate memory */
masterDataSend = (uint8_t*)calloc(100, sizeof(uint8_t));
masterDataReceive = (uint8_t*)calloc(100, sizeof(uint8_t));
slaveDataSend = (uint8_t*)calloc(100, sizeof(uint8_t));
slaveDataReceive = (uint8_t*)calloc(100, sizeof(uint8_t));
bufferSize = 100U;
testStatus[0] = true;
LPSPI_DRV_SlaveTransfer(0U, slaveDataSend,
    slaveDataReceive, bufferSize);
LPSPI_DRV_MasterTransferBlocking(1U, &Send_MasterConfig0, masterDataSend,
    masterDataReceive, bufferSize, TIMEOUT);
```

### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\drivers\src\lpspi\lpspi_irq.c
${S32SDK_PATH}\platform\drivers\src\lpspi\lpspi_master_driver.c
${S32SDK_PATH}\platform\drivers\src\lpspi\lpspi_shared_function.c
${S32SDK_PATH}\platform\drivers\src\lpspi\lpspi_slave_driver.c
${S32SDK_PATH}\platform\drivers\src\lpspi\lpspi_hw_access.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc
${S32SDK_PATH}\platform\drivers\src\lpspi
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

[Clock Manager OS Interface \(OSIF\)](#) [Interrupt Manager \(Interrupt\)](#) [Enhanced Direct Memory Access \(eDMA\)](#)

## Modules

- [LPSPI Driver](#)

*Low Power Serial Peripheral Interface Peripheral Driver.*

## 16.88 Low Power Timer (LPTMR)

### 16.88.1 Detailed Description

The S32 SDK provides Peripheral Drivers for the Low Power Timer (LPTMR) module of S32 SDK devices.

The LPTMR is a configurable 16-bit counter that can operate in two functional modes:

- Timer mode with selectable prescaler and clock source (periodic or free-running).
- Pulse-Counter mode, with configurable glitch filter, that can count events (internal or external)

### Modules

- [LPTMR Driver](#)

*Low Power Timer Peripheral Driver.*

## 16.89 Low Power Universal Asynchronous Receiver-Transmitter (LPUART)

### 16.89.1 Detailed Description

The S32 SDK provides a Peripheral Driver for the Low Power Universal Asynchronous Receiver-Transmitter (LP↔UART) module of S32 SDK devices.

The LPUART module is used for serial communication, supporting LIN master and slave operation. These sections describe the S32 SDK software modules API that can be used for initializing and configuring the module, as well as initiating serial communications using the interrupt-based method.

#### Modules

- [LPUART Driver](#)

*This module covers the functionality of the Low Power Universal Asynchronous Receiver-Transmitter (LPUART) peripheral driver.*

## 16.90 Low level API

### 16.90.1 Detailed Description

Low level layer consists of functions that call LIN driver API.

This layer contains the implementation of LIN hardware initialization and deinitialization, getting LIN node's current state, sending wakeup signals, enabling and disabling interrupts, sending frame data from a buffer, receiving frame data into a buffer, handling timeout and callbacks from LIN driver.

#### Data Structures

- struct [lin\\_word\\_status\\_str\\_t](#)  
*status of LIN bus Implements : [lin\\_word\\_status\\_str\\_t\\_Class](#) [More...](#)*
- struct [lin\\_serial\\_number\\_t](#)  
*Serial number Implements : [lin\\_serial\\_number\\_t\\_Class](#). [More...](#)*
- struct [lin\\_node\\_attribute\\_t](#)  
*Attributes of LIN node Implements : [lin\\_node\\_attribute\\_t\\_Class](#). [More...](#)*
- struct [lin\\_associate\\_frame\\_t](#)  
*Informations of associated frame Implements : [lin\\_associate\\_frame\\_t\\_Class](#). [More...](#)*
- struct [lin\\_frame\\_t](#)  
*Frame description structure Implements : [lin\\_frame\\_t\\_Class](#). [More...](#)*
- struct [lin\\_schedule\\_data\\_t](#)  
*LIN schedule structure Implements : [lin\\_schedule\\_data\\_t\\_Class](#). [More...](#)*
- struct [lin\\_schedule\\_t](#)  
*Schedule table description Implements : [lin\\_schedule\\_t\\_Class](#). [More...](#)*
- struct [lin\\_transport\\_layer\\_queue\\_t](#)  
*Transport layer queue Implements : [lin\\_transport\\_layer\\_queue\\_t\\_Class](#). [More...](#)*
- struct [lin\\_tl\\_descriptor\\_t](#)  
*Transport layer description Implements : [lin\\_tl\\_descriptor\\_t\\_Class](#). [More...](#)*
- struct [lin\\_protocol\\_user\\_config\\_t](#)  
*Configuration structure Implements : [lin\\_protocol\\_user\\_config\\_t\\_Class](#). [More...](#)*
- struct [lin\\_master\\_data\\_t](#)  
*LIN master configuration structure Implements : [lin\\_master\\_data\\_t\\_Class](#). [More...](#)*
- struct [lin\\_protocol\\_state\\_t](#)  
*LIN protocol status structure Implements : [lin\\_protocol\\_state\\_t\\_Class](#). [More...](#)*

#### Macros

- `#define SERVICE\_ASSIGN\_NAD 0xB0U`
- `#define SERVICE\_ASSIGN\_FRAME\_ID 0xB1U`
- `#define SERVICE\_READ\_BY\_IDENTIFY 0xB2U`
- `#define SERVICE\_CONDITIONAL\_CHANGE\_NAD 0xB3U`
- `#define SERVICE\_SAVE\_CONFIGURATION 0xB6U`
- `#define SERVICE\_ASSIGN\_FRAME\_ID\_RANGE 0xB7U`
- `#define SERVICE\_READ\_DATA\_BY\_IDENTIFY 0x22U`
- `#define SERVICE\_WRITE\_DATA\_BY\_IDENTIFY 0x2EU`
- `#define SERVICE\_SESSION\_CONTROL 0x10U`
- `#define SERVICE\_IO\_CONTROL\_BY\_IDENTIFY 0x2FU`
- `#define SERVICE\_FAULT\_MEMORY\_READ 0x19U`
- `#define SERIVCE\_FAULT\_MEMORY\_CLEAR 0x14U`
- `#define PCI\_SAVE\_CONFIGURATION 0x01U`
- `#define PCI\_RES\_READ\_BY\_IDENTIFY 0x06U`

- #define `PCI_RES_SAVE_CONFIGURATION` 0x01U
  - #define `PCI_RES_ASSIGN_FRAME_ID_RANGE` 0x01U
  - #define `LIN_READ_USR_DEF_MIN` 32U
  - #define `LIN_READ_USR_DEF_MAX` 63U
  - #define `LD_NEGATIVE_RESPONSE` 0x53U
  - #define `LD_POSITIVE_RESPONSE` 0x54U
  - #define `LIN_LLD_OK` 0x00U
  - #define `LIN_LLD_ERROR` 0xFFU
  - #define `LIN_SLAVE` 0
- Mode of LIN node (master or slave)*
- #define `LIN_MASTER` 1
  - #define `LIN_TL_CALLBACK_HANDLER`(iii, tl\_event\_id, id) `lin_tl_callback_handler`((iii), (tl\_event\_id), (id))
  - #define `CALLBACK_HANDLER`(iii, event\_id, id) `lin_pid_resp_callback_handler`((iii), (event\_id), (id))
- `CALLBACK_HANDLER.`

### Typedefs

- typedef `l_u8 lin_tl_pdu_data_t`[8]  
*PDU data. Implements : `lin_tl_pdu_data_t_Class`.*
- typedef `l_u8 lin_tl_queue_t`[8]  
*LIN transport layer queue Implements : `lin_tl_queue_t_Class`.*

### Enumerations

- enum `lin_llid_event_id_t` {  
`LIN_LLD_PID_OK` = 0x00U, `LIN_LLD_TX_COMPLETED` = 0x01U, `LIN_LLD_RX_COMPLETED` = 0x02U,  
`LIN_LLD_PID_ERR` = 0x03U,  
`LIN_LLD_FRAME_ERR` = 0x04U, `LIN_LLD_CHECKSUM_ERR` = 0x05U, `LIN_LLD_READBACK_ERR` =  
0x06U, `LIN_LLD_NODATA_TIMEOUT` = 0x07U,  
`LIN_LLD_BUS_ACTIVITY_TIMEOUT` = 0x08U }  
*Event id Implements : `lin_llid_event_id_t_Class`.*
- enum `lin_protocol_handle_t` { `LIN_PROTOCOL_21` = 0x00U, `LIN_PROTOCOL_J2602` = 0x01U, `LIN_PROTOCOL_13` = 0x02U, `LIN_PROTOCOL_20` = 0x03U }  
*List of protocols Implements : `lin_protocol_handle_t_Class`.*
- enum `lin_diagnostic_class_t` { `LIN_DIAGNOSTIC_CLASS_I` = 0x01U, `LIN_DIAGNOSTIC_CLASS_II` =  
0x02U, `LIN_DIAGNOSTIC_CLASS_III` = 0x03U }  
*List of diagnostic classes Implements : `lin_diagnostic_class_t_Class`.*
- enum `lin_frame_type_t` { `LIN_FRM_UNCD` = 0x00U, `LIN_FRM_EVNT` = 0x01U, `LIN_FRM_SPRDC` = 0x10U,  
`LIN_FRM_DIAG` = 0x11U }  
*Types of frame Implements : `lin_frame_type_t_Class`.*
- enum `lin_frame_response_t` { `LIN_RES_PUB` = 0x00U, `LIN_RES_SUB` = 0x01U }  
*LIN frame response Implements : `lin_frame_response_t_Class`.*
- enum `lin_sch_tbl_type_t` {  
`LIN_SCH_TBL_NULL` = 0x00U, `LIN_SCH_TBL_NORM` = 0x01U, `LIN_SCH_TBL_DIAG` = 0x02U, `LIN_SCH_TBL_GO_TO_SLEEP` = 0x03U,  
`LIN_SCH_TBL_COLL_RESOLV` = 0x04U }  
*Types of schedule tables Implements : `lin_sch_tbl_type_t_Class`.*
- enum `l_diagnostic_mode_t` { `DIAG_NONE` = 0x00U, `DIAG_INTERLEAVE_MODE` = 0x01U, `DIAG_ONLY_MODE` = 0x02U }  
*Diagnostic mode Implements : `l_diagnostic_mode_t_Class`.*
- enum `lin_service_status_t` { `LD_SERVICE_BUSY` = 0x00U, `LD_REQUEST_FINISHED` = 0x01U, `LD_SERVICE_IDLE` = 0x02U, `LD_SERVICE_ERROR` = 0x03U }  
*Status of the last configuration call for LIN 2.1 Implements : `lin_service_status_t_Class`.*

- enum `lin_last_cfg_result_t` { `LD_SUCCESS` = 0x00U, `LD_NEGATIVE` = 0x01U, `LD_NO_RESPONSE` = 0x02U, `LD_OVERWRITTEN` = 0x03U }  
*Status of the last configuration call completed Implements : `lin_last_cfg_result_t` Class.*
- enum `lin_tl_event_id_t` {  
`TL_MAKE_RES_DATA` = 0x00U, `TL_SLAVE_GET_ACTION` = 0x01U, `TL_TX_COMPLETED` = 0x02U, `TL_RX_COMPLETED` = 0x03U,  
`TL_ERROR` = 0x04U, `TL_TIMEOUT_SERVICE` = 0x05U, `TL_HANDLER_INTERLEAVE_MODE` = 0x06U,  
`TL_RECEIVE_MESSAGE` = 0x07U }  
*Transport layer event IDs Implements : `lin_tl_event_id_t` Class.*
- enum `lin_tl_callback_return_t` { `TL_ACTION_NONE` = 0x00U, `TL_ACTION_ID_IGNORE` = 0x01U }  
*Transport layer event IDs Implements : `lin_tl_callback_return_t` Class.*
- enum `ld_queue_status_t` {  
`LD_NO_DATA` = 0x00U, `LD_DATA_AVAILABLE` = 0x01U, `LD_RECEIVE_ERROR` = 0x02U, `LD_QUEUE_FULL` = 0x03U,  
`LD_QUEUE_AVAILABLE` = 0x04U, `LD_QUEUE_EMPTY` = 0x05U, `LD_TRANSMIT_ERROR` = 0x06U, `LD_TRANSFER_ERROR` = 0x07U }  
*Status of queue Implements : `ld_queue_status_t` Class.*
- enum `lin_message_status_t` {  
`LD_NO_MSG` = 0x00U, `LD_IN_PROGRESS` = 0x01U, `LD_COMPLETED` = 0x02U, `LD_FAILED` = 0x03U,  
`LD_N_AS_TIMEOUT` = 0x04U, `LD_N_CR_TIMEOUT` = 0x05U, `LD_WRONG_SN` = 0x06U }  
*Status of LIN message Implements : `lin_message_status_t` Class.*
- enum `lin_diagnostic_state_t` {  
`LD_DIAG_IDLE` = 0x01U, `LD_DIAG_TX_PHY` = 0x02U, `LD_DIAG_TX_FUNCTIONAL` = 0x03U, `LD_DIAG_TX_INTERLEAVED` = 0x04U,  
`LD_DIAG_RX_PHY` = 0x05U, `LD_DIAG_RX_FUNCTIONAL` = 0x06U, `LD_DIAG_RX_INTERLEAVED` = 0x07U }  
*LIN diagnostic state Implements : `lin_diagnostic_state_t` Class.*
- enum `lin_message_timeout_type_t` { `LD_NO_CHECK_TIMEOUT` = 0x00U, `LD_CHECK_N_AS_TIMEOUT` = 0x01U, `LD_CHECK_N_CR_TIMEOUT` = 0x02U }  
*Types of message timeout Implements : `lin_message_timeout_type_t` Class.*
- enum `diag_interleaved_state_t` { `DIAG_NOT_START` = 0x00U, `DIAG_NO_RESPONSE` = 0x01U, `DIAG_RESPONSE` = 0x02U }  
*State of diagnostic interleaved mode Implements : `diag_interleaved_state_t` Class.*

## Functions

- `lin_tl_callback_return_t lin_tl_callback_handler` (l\_ifc\_handle iii, `lin_tl_event_id_t` tl\_event\_id, l\_u8 id)
- `l_u8 ld_read_by_id_callout` (l\_ifc\_handle iii, l\_u8 id, l\_u8 \*data)
- static `l_u16 lin_calc_max_header_timeout_cnt` (l\_u32 baudRate)  
*Computes maximum header timeout.*
- static `l_u16 lin_calc_max_res_timeout_cnt` (l\_u32 baudRate, l\_u8 size)  
*Computes the maximum response timeout.*
- `l_u8 lin_process_parity` (l\_u8 pid, l\_u8 typeAction)  
*Makes or checks parity bits. If action is checking parity, the function returns ID value if parity bits are correct or 0xFF if parity bits are incorrect. If action is making parity bits, then from input value of ID, the function returns PID.*
- void `lin_pid_resp_callback_handler` (l\_ifc\_handle iii, const `lin_lld_event_id_t` event\_id, l\_u8 id)  
*Callback handler for low level events.*
- `l_bool lin_lld_init` (l\_ifc\_handle iii)  
*This function initializes a LIN hardware instance for operation. This function will initialize the run-time state structure to keep track of the on-going transfers, initialize the module to user defined settings and default settings, configure the IRQ state structure and enable the module-level interrupt to the core, and enable the LIN hardware module transmitter and receiver.*
- void `lin_lld_deinit` (l\_ifc\_handle iii)  
*This function disconnect the node from the cluster and free all hardware used.*



- `I_u8 lin_lld_int_enable (I_ifc_handle iii)`  
*Enable the interrupt related to the interface.*
- `I_u8 lin_lld_int_disable (I_ifc_handle iii)`  
*Disable the interrupt related to the interface.*
- `lin_node_state_t lin_lld_get_state (I_ifc_handle iii)`  
*This function gets current state of an interface.*
- `I_u8 lin_lld_tx_header (I_ifc_handle iii, I_u8 id)`  
*This function sends frame header for the input PID.*
- `I_u8 lin_lld_tx_wake_up (I_ifc_handle iii)`  
*This function send a wakeup signal.*
- `I_u8 lin_lld_ignore_response (I_ifc_handle iii)`  
*This function terminates an on-going data transmission/reception.*
- `I_u8 lin_lld_set_low_power_mode (I_ifc_handle iii)`  
*Let the low level driver go to low power mode.*
- `I_u8 lin_lld_set_response (I_ifc_handle iii, I_u8 response_length)`  
*This function sends frame data that is contained in LIN\_lld\_response\_buffer[iii].*
- `I_u8 lin_lld_rx_response (I_ifc_handle iii, I_u8 response_length)`  
*This function receives frame data into the LIN\_lld\_response\_buffer[iii] buffer.*
- `void lin_lld_timeout_service (I_ifc_handle iii)`  
*Callback function for Timer Interrupt Handler In timer IRQ handler, call this function. Used to check if frame timeout has occurred during frame data transmission and reception, to check for N\_As and N\_Cr timeout for LIN 2.1 and above. This function also check if there is no LIN bus communication (no headers and no frame data transferring) for Idle timeout (s), then put LIN node to Sleep mode. Users may initialize a timer (for example FTM)with period of Timeout unit (default: 500 micro seconds) to call `lin_lld_timeout_service()`. For an interface iii, Idle timeout (s) =  $\text{max\_idle\_timeout\_cnt} * \text{Timeout unit (us)}$  frame timeout (us) =  $\text{frame\_timeout\_cnt} * \text{Timeout unit (us)}$  N\_As timeout (us) =  $\text{N\_As\_timeout} * \text{Timeout unit (us)}$  N\_Cr timeout (us) =  $\text{N\_Cr\_timeout} * \text{Timeout unit (us)}$*

## Variables

- `const lin_node_attribute_t g_lin_node_attribute_array [LIN_NUM_OF_SLAVE_IFCS]`
- `lin_master_data_t g_lin_master_data_array [LIN_NUM_OF_MASTER_IFCS]`
- `lin_tl_descriptor_t g_lin_tl_descriptor_array [LIN_NUM_OF_IFCS]`
- `const lin_protocol_user_config_t g_lin_protocol_user_cfg_array [LIN_NUM_OF_IFCS]`
- `lin_protocol_state_t g_lin_protocol_state_array [LIN_NUM_OF_IFCS]`
- `volatile I_u8 g_lin_frame_data_buffer [LIN_FRAME_BUF_SIZE]`
- `volatile I_u8 g_lin_flag_handle_tbl [LIN_FLAG_BUF_SIZE]`
- `volatile I_bool g_lin_frame_flag_handle_tbl [LIN_NUM_OF_FRMS]`
- `const I_u32 g_lin_virtual_ifc [LIN_NUM_OF_IFCS]`
- `const I_ifc_handle g_lin_hardware_ifc [HARDWARE_INSTANCE_COUNT]`
- `const lin_timer_get_time_interval_t timerGetTimeIntervalCallbackArr [LIN_NUM_OF_IFCS]`
- `volatile I_u8 g_buffer_backup_data [8]`
- `volatile I_u8 g_lin_frame_updating_flag_tbl [LIN_NUM_OF_FRMS]`

## 16.90.2 Data Structure Documentation

### 16.90.2.1 struct lin\_word\_status\_str\_t

status of LIN bus Implements : `lin_word_status_str_t_Class`

Definition at line 148 of file `lin.h`.

## Data Fields

- unsigned int [error\\_in\\_res](#): 1
- unsigned int [successful\\_transfer](#): 1
- unsigned int [overrun](#): 1
- unsigned int [go\\_to\\_sleep\\_flg](#): 1
- unsigned int [bus\\_activity](#): 1
- unsigned int [event\\_trigger\\_collision\\_flg](#): 1
- unsigned int [save\\_config\\_flg](#): 1
- unsigned int [reserved](#): 1
- unsigned int [last\\_pid](#): 8

## Field Documentation

### 16.90.2.1.1 unsigned int bus\_activity

Bus activity

Definition at line 154 of file lin.h.

### 16.90.2.1.2 unsigned int error\_in\_res

Error in response

Definition at line 150 of file lin.h.

### 16.90.2.1.3 unsigned int event\_trigger\_collision\_flg

Event trigger collision

Definition at line 155 of file lin.h.

### 16.90.2.1.4 unsigned int go\_to\_sleep\_flg

Goto sleep

Definition at line 153 of file lin.h.

### 16.90.2.1.5 unsigned int last\_pid

Last PID

Definition at line 158 of file lin.h.

### 16.90.2.1.6 unsigned int overrun

Overrun

Definition at line 152 of file lin.h.

### 16.90.2.1.7 unsigned int reserved

Dummy

Definition at line 157 of file lin.h.

### 16.90.2.1.8 unsigned int save\_config\_flg

Save configuration

Definition at line 156 of file lin.h.

### 16.90.2.1.9 unsigned int successful\_transfer

Successful transfer

Definition at line 151 of file lin.h.

#### 16.90.2.2 struct lin\_serial\_number\_t

Serial number Implements : lin\_serial\_number\_t\_Class.

Definition at line 175 of file lin.h.

##### Data Fields

- [l\\_u8 serial\\_0](#)
- [l\\_u8 serial\\_1](#)
- [l\\_u8 serial\\_2](#)
- [l\\_u8 serial\\_3](#)

##### Field Documentation

#### 16.90.2.2.1 l\_u8 serial\_0

Serial 0

Definition at line 177 of file lin.h.

#### 16.90.2.2.2 l\_u8 serial\_1

Serial 1

Definition at line 178 of file lin.h.

#### 16.90.2.2.3 l\_u8 serial\_2

Serial 2

Definition at line 179 of file lin.h.

#### 16.90.2.2.4 l\_u8 serial\_3

Serial 3

Definition at line 180 of file lin.h.

#### 16.90.2.3 struct lin\_node\_attribute\_t

Attributes of LIN node Implements : lin\_node\_attribute\_t\_Class.

Definition at line 187 of file lin.h.

##### Data Fields

- [l\\_u8 \\* configured\\_NAD\\_ptr](#)
- [l\\_u8 initial\\_NAD](#)
- [lin\\_product\\_id\\_t product\\_id](#)
- [lin\\_serial\\_number\\_t serial\\_number](#)
- [l\\_u8 \\* resp\\_err\\_frm\\_id\\_ptr](#)
- [l\\_u8 num\\_frame\\_have\\_esignal](#)
- [l\\_signal\\_handle response\\_error](#)
- [l\\_u16 \\* response\\_error\\_byte\\_offset\\_ptr](#)
- [l\\_u8 \\* response\\_error\\_bit\\_offset\\_ptr](#)
- [l\\_u8 num\\_of\\_fault\\_state\\_signal](#)
- [const l\\_signal\\_handle \\* fault\\_state\\_signal\\_ptr](#)
- [l\\_u16 P2\\_min](#)
- [l\\_u16 ST\\_min](#)
- [l\\_u16 N\\_As\\_timeout](#)

- `I_u16 N_Cr_timeout`
- `I_u8 number_support_sid`
- `const I_u8 * service_supported_ptr`
- `I_u8 * service_flags_ptr`

## Field Documentation

### 16.90.2.3.1 `I_u8* configured_NAD_ptr`

NAD value used in configuration command

Definition at line 189 of file `lin.h`.

### 16.90.2.3.2 `const I_signal_handle* fault_state_signal_ptr`

List of fault state signal

Definition at line 199 of file `lin.h`.

### 16.90.2.3.3 `I_u8 initial_NAD`

Initial NAD

Definition at line 190 of file `lin.h`.

### 16.90.2.3.4 `I_u16 N_As_timeout`

`N_As_timeout`

Definition at line 202 of file `lin.h`.

### 16.90.2.3.5 `I_u16 N_Cr_timeout`

`N_Cr_timeout`

Definition at line 203 of file `lin.h`.

### 16.90.2.3.6 `I_u8 num_frame_have_esignal`

Number of frame contain error signal

Definition at line 194 of file `lin.h`.

### 16.90.2.3.7 `I_u8 num_of_fault_state_signal`

Number of Fault state signal

Definition at line 198 of file `lin.h`.

### 16.90.2.3.8 `I_u8 number_support_sid`

Number of supported diagnostic services

Definition at line 204 of file `lin.h`.

### 16.90.2.3.9 `I_u16 P2_min`

`P2_min`

Definition at line 200 of file `lin.h`.

### 16.90.2.3.10 `lin_product_id_t product_id`

Product ID

Definition at line 191 of file `lin.h`.

**16.90.2.3.11** `I_u8* resp_err_frm_id_ptr`

List index of frame contain response error signal

Definition at line 193 of file lin.h.

**16.90.2.3.12** `I_signal_handle response_error`

Signal used to update response error

Definition at line 195 of file lin.h.

**16.90.2.3.13** `I_u8* response_error_bit_offset_ptr`

Bit offset of response error signal

Definition at line 197 of file lin.h.

**16.90.2.3.14** `I_u16* response_error_byte_offset_ptr`

Byte offset of response error signal

Definition at line 196 of file lin.h.

**16.90.2.3.15** `lin_serial_number_t serial_number`

Serial number

Definition at line 192 of file lin.h.

**16.90.2.3.16** `I_u8* service_flags_ptr`

List of associated flags with supported diagnostic services

Definition at line 206 of file lin.h.

**16.90.2.3.17** `const I_u8* service_supported_ptr`

List of supported diagnostic service

Definition at line 205 of file lin.h.

**16.90.2.3.18** `I_u16 ST_min`

ST min

Definition at line 201 of file lin.h.

**16.90.2.4** `struct lin_associate_frame_t`

Informations of associated frame Implements : `lin_associate_frame_t_Class`.

Definition at line 238 of file lin.h.

**Data Fields**

- `I_u8 num_of_associated_uncond_frames`
- `const I_frame_handle * associated_uncond_frame_ptr`
- `I_u8 coll_resolv_schd`

**Field Documentation****16.90.2.4.1** `const I_frame_handle* associated_uncond_frame_ptr`

Associated unconditional frame ID

Definition at line 241 of file lin.h.

#### 16.90.2.4.2 `I_u8 coll_resolv_schd`

Collision resolver index in the schedule table, used in event trigger frame case MASTER

Definition at line 242 of file lin.h.

#### 16.90.2.4.3 `I_u8 num_of_associated_uncond_frames`

Number of associated unconditional frame ID

Definition at line 240 of file lin.h.

#### 16.90.2.5 `struct lin_frame_t`

Frame description structure Implements : `lin_frame_t_Class`.

Definition at line 249 of file lin.h.

##### Data Fields

- [lin\\_frame\\_type\\_t frm\\_type](#)
- [I\\_u8 frm\\_len](#)
- [lin\\_frame\\_response\\_t frm\\_response](#)
- [I\\_u16 frm\\_offset](#)
- [I\\_u16 flag\\_offset](#)
- [I\\_u8 flag\\_size](#)
- `const lin\_associate\_frame\_t * frame_data_ptr`

##### Field Documentation

#### 16.90.2.5.1 `I_u16 flag_offset`

Flag byte offset in flag buffer

Definition at line 255 of file lin.h.

#### 16.90.2.5.2 `I_u8 flag_size`

Flag size in flag buffer

Definition at line 256 of file lin.h.

#### 16.90.2.5.3 `const lin_associate_frame_t* frame_data_ptr`

List of Signal to which the frame is associated and its offset

Definition at line 257 of file lin.h.

#### 16.90.2.5.4 `I_u8 frm_len`

Length of the frame

Definition at line 252 of file lin.h.

#### 16.90.2.5.5 `I_u16 frm_offset`

Frame byte offset in frame buffer

Definition at line 254 of file lin.h.

#### 16.90.2.5.6 `lin_frame_response_t frm_response`

Action response when received PID

Definition at line 253 of file lin.h.

**16.90.2.5.7 lin\_frame\_type\_t frm\_type**

Frame information (unconditional or event triggered..)

Definition at line 251 of file lin.h.

**16.90.2.6 struct lin\_schedule\_data\_t**

LIN schedule structure Implements : lin\_schedule\_data\_t\_Class.

Definition at line 286 of file lin.h.

**Data Fields**

- I\_frame\_handle [frm\\_id](#)
- I\_u8 [delay\\_integer](#)
- [lin\\_tl\\_queue\\_t](#) [tl\\_queue\\_data](#)

**Field Documentation****16.90.2.6.1 I\_u8 delay\_integer**

Actual slot time in INTEGER for one frame

Definition at line 289 of file lin.h.

**16.90.2.6.2 I\_frame\_handle frm\_id**

Frame ID, in case of unconditional or event triggered frame. For sporadic frame the value will be 0 (zero)

Definition at line 288 of file lin.h.

**16.90.2.6.3 lin\_tl\_queue\_t tl\_queue\_data**

Data used in case of diagnostic or configuration frame

Definition at line 290 of file lin.h.

**16.90.2.7 struct lin\_schedule\_t**

Schedule table description Implements : lin\_schedule\_t\_Class.

Definition at line 297 of file lin.h.

**Data Fields**

- I\_u8 [num\\_slots](#)
- [lin\\_sch\\_tbl\\_type\\_t](#) [sch\\_tbl\\_type](#)
- const [lin\\_schedule\\_data\\_t](#) \* [ptr\\_sch\\_data\\_ptr](#)

**Field Documentation****16.90.2.7.1 I\_u8 num\_slots**

Number of frame slots in the schedule table

Definition at line 299 of file lin.h.

**16.90.2.7.2 const lin\_schedule\_data\_t\* ptr\_sch\_data\_ptr**

Address of the schedule table

Definition at line 301 of file lin.h.

### 16.90.2.7.3 `lin_sch_tbl_type_t` `sch_tbl_type`

Schedule table type

Definition at line 300 of file `lin.h`.

### 16.90.2.8 `struct lin_transport_layer_queue_t`

Transport layer queue Implements : `lin_transport_layer_queue_t_Class`.

Definition at line 435 of file `lin.h`.

#### Data Fields

- `I_u16 queue_header`
- `I_u16 queue_tail`
- `Id_queue_status_t queue_status`
- `I_u16 queue_current_size`
- `I_u16 queue_max_size`
- `lin_tl_pdu_data_t * tl_pdu_ptr`

#### Field Documentation

#### 16.90.2.8.1 `I_u16 queue_current_size`

Current size

Definition at line 440 of file `lin.h`.

#### 16.90.2.8.2 `I_u16 queue_header`

The first element of queue

Definition at line 437 of file `lin.h`.

#### 16.90.2.8.3 `I_u16 queue_max_size`

Maximum size

Definition at line 441 of file `lin.h`.

#### 16.90.2.8.4 `Id_queue_status_t queue_status`

Status of queue

Definition at line 439 of file `lin.h`.

#### 16.90.2.8.5 `I_u16 queue_tail`

The last element of queue

Definition at line 438 of file `lin.h`.

#### 16.90.2.8.6 `lin_tl_pdu_data_t* tl_pdu_ptr`

PDU data

Definition at line 442 of file `lin.h`.

### 16.90.2.9 `struct lin_tl_descriptor_t`

Transport layer description Implements : `lin_tl_descriptor_t_Class`.

Definition at line 461 of file `lin.h`.



## Data Fields

- [lin\\_transport\\_layer\\_queue\\_t tl\\_tx\\_queue](#)
- [lin\\_transport\\_layer\\_queue\\_t tl\\_rx\\_queue](#)
- [lin\\_message\\_status\\_t rx\\_msg\\_status](#)
- [l\\_u16 rx\\_msg\\_size](#)
- [lin\\_message\\_status\\_t tx\\_msg\\_status](#)
- [l\\_u16 tx\\_msg\\_size](#)
- [lin\\_last\\_cfg\\_result\\_t last\\_cfg\\_result](#)
- [l\\_u8 last\\_RSID](#)
- [l\\_u8 ld\\_error\\_code](#)
- [lin\\_message\\_timeout\\_type\\_t check\\_timeout\\_type](#)
- [l\\_u16 check\\_timeout](#)
- [lin\\_product\\_id\\_t \\* product\\_id\\_ptr](#)
- [l\\_u8 num\\_of\\_pdu](#)
- [l\\_u8 frame\\_counter](#)
- [lin\\_diagnostic\\_state\\_t diag\\_state](#)
- [diag\\_interleaved\\_state\\_t diag\\_interleave\\_state](#)
- [l\\_u16 interleave\\_timeout\\_counter](#)
- [l\\_u8 slave\\_resp\\_cnt](#)
- [lin\\_service\\_status\\_t service\\_status](#)
- [bool ld\\_return\\_data](#)
- [bool FF\\_pdu\\_received](#)
- [l\\_u8 \\* receive\\_message\\_ptr](#)
- [l\\_u8 \\* receive\\_NAD\\_ptr](#)
- [l\\_u16 \\* receive\\_message\\_length\\_ptr](#)

## Field Documentation

16.90.2.9.1 [l\\_u16 check\\_timeout](#)

Timeout counter for N\_As and N\_Cr timeout

Definition at line 481 of file lin.h.

16.90.2.9.2 [lin\\_message\\_timeout\\_type\\_t check\\_timeout\\_type](#)

Timeout type

Definition at line 480 of file lin.h.

16.90.2.9.3 [diag\\_interleaved\\_state\\_t diag\\_interleave\\_state](#)

state of diagnostic interleaved mode

Definition at line 486 of file lin.h.

16.90.2.9.4 [lin\\_diagnostic\\_state\\_t diag\\_state](#)

Diagnostic state

Definition at line 485 of file lin.h.

16.90.2.9.5 [bool FF\\_pdu\\_received](#)

Status of FF pdu

Definition at line 492 of file lin.h.

**16.90.2.9.6   I\_u8 frame\_counter**

Frame counter in received message

Definition at line 484 of file lin.h.

**16.90.2.9.7   I\_u16 interleave\_timeout\_counter**

Interleaved timeout counter

Definition at line 487 of file lin.h.

**16.90.2.9.8   lin\_last\_cfg\_result\_t last\_cfg\_result**

Status of the last configuration service

Definition at line 476 of file lin.h.

**16.90.2.9.9   I\_u8 last\_RSID**

RSID of the last node configuration service

Definition at line 477 of file lin.h.

**16.90.2.9.10   I\_u8 ld\_error\_code**

Error code in case of positive response

Definition at line 478 of file lin.h.

**16.90.2.9.11   bool ld\_return\_data**

Decide return data of diagnostic frame to pointer of ld\_receive\_message function

Definition at line 491 of file lin.h.

**16.90.2.9.12   I\_u8 num\_of\_pdu**

Number of received pdu

Definition at line 483 of file lin.h.

**16.90.2.9.13   lin\_product\_id\_t\* product\_id\_ptr**

To store address of RAM area contain response

Definition at line 482 of file lin.h.

**16.90.2.9.14   I\_u16\* receive\_message\_length\_ptr**

Pointer to receive\_message\_length of user

Definition at line 497 of file lin.h.

**16.90.2.9.15   I\_u8\* receive\_message\_ptr**

Pointer to receive\_message array of user

Definition at line 495 of file lin.h.

**16.90.2.9.16   I\_u8\* receive\_NAD\_ptr**

Pointer to receive\_NAD of user

Definition at line 496 of file lin.h.

**16.90.2.9.17** `_u16 rx_msg_size`

Size of message in queue

Definition at line 470 of file lin.h.

**16.90.2.9.18** `lin_message_status_t rx_msg_status`

Cooked rx status

Definition at line 469 of file lin.h.

**16.90.2.9.19** `lin_service_status_t service_status`

Status of the last configuration service

Definition at line 489 of file lin.h.

**16.90.2.9.20** `_u8 slave_resp_cnt`

Slave Response data counter

Definition at line 488 of file lin.h.

**16.90.2.9.21** `lin_transport_layer_queue_t tl_rx_queue`

Pointer to receive queue on TL

Definition at line 465 of file lin.h.

**16.90.2.9.22** `lin_transport_layer_queue_t tl_tx_queue`

Pointer to transmit queue on TL

Definition at line 464 of file lin.h.

**16.90.2.9.23** `_u16 tx_msg_size`

Size of message in queue

Definition at line 474 of file lin.h.

**16.90.2.9.24** `lin_message_status_t tx_msg_status`

Cooked tx status

Definition at line 473 of file lin.h.

**16.90.2.10** `struct lin_protocol_user_config_t`

Configuration structure Implements : `lin_protocol_user_config_t_Class`.

Definition at line 507 of file lin.h.

**Data Fields**

- [lin\\_protocol\\_handle\\_t protocol\\_version](#)
- [lin\\_protocol\\_handle\\_t language\\_version](#)
- [lin\\_diagnostic\\_class\\_t diagnostic\\_class](#)
- `bool function`
- `_u8 number_of_configurable_frames`
- `_u8 frame_start`
- `const lin_frame_t * frame_tbl_ptr`
- `const _u16 * list_identifiers_ROM_ptr`
- `_u8 * list_identifiers_RAM_ptr`

- `I_u16 max_idle_timeout_cnt`
- `I_u8 num_of_schedules`
- `I_u8 schedule_start`
- `const lin_schedule_t * schedule_tbl`
- `I_ifc_slave_handle slave_ifc_handle`
- `I_ifc_master_handle master_ifc_handle`
- `lin_user_config_t * lin_user_config_ptr`
- `lin_tl_pdu_data_t * tl_tx_queue_data_ptr`
- `lin_tl_pdu_data_t * tl_rx_queue_data_ptr`
- `I_u16 max_message_length`

## Field Documentation

### 16.90.2.10.1 `lin_diagnostic_class_t` `diagnostic_class`

Diagnostic class

Definition at line 511 of file `lin.h`.

### 16.90.2.10.2 `I_u8` `frame_start`

Start index of frame list

Definition at line 515 of file `lin.h`.

### 16.90.2.10.3 `const lin_frame_t*` `frame_tbl_ptr`

Frame list except diagnostic frames

Definition at line 516 of file `lin.h`.

### 16.90.2.10.4 `bool` `function`

Function `LIN_MASTER` or `LIN_SLAVE_`)

Definition at line 512 of file `lin.h`.

### 16.90.2.10.5 `lin_protocol_handle_t` `language_version`

Language version

Definition at line 510 of file `lin.h`.

### 16.90.2.10.6 `lin_user_config_t*` `lin_user_config_ptr`

Pointer to LIN driver user configuration structure

Definition at line 526 of file `lin.h`.

### 16.90.2.10.7 `I_u8*` `list_identifiers_RAM_ptr`

Configuration in RAM

Definition at line 519 of file `lin.h`.

### 16.90.2.10.8 `const I_u16*` `list_identifiers_ROM_ptr`

Configuration in ROM

Definition at line 518 of file `lin.h`.

### 16.90.2.10.9 `I_ifc_master_handle` `master_ifc_handle`

Interface handler of master node

Definition at line 525 of file lin.h.

**16.90.2.10.10** `l_u16 max_idle_timeout_cnt`

Max Idle timeout counter

Definition at line 520 of file lin.h.

**16.90.2.10.11** `l_u16 max_message_length`

Max message length

Definition at line 530 of file lin.h.

**16.90.2.10.12** `l_u8 num_of_schedules`

Number of schedule table

Definition at line 521 of file lin.h.

**16.90.2.10.13** `l_u8 number_of_configurable_frames`

Number of frame except diagnostic frames

Definition at line 514 of file lin.h.

**16.90.2.10.14** `lin_protocol_handle_t protocol_version`

Protocol version

Definition at line 509 of file lin.h.

**16.90.2.10.15** `l_u8 schedule_start`

Start index of schedule table list

Definition at line 522 of file lin.h.

**16.90.2.10.16** `const lin_schedule_t* schedule_tbl`

Schedule table list

Definition at line 523 of file lin.h.

**16.90.2.10.17** `l_ifc_slave_handle slave_ifc_handle`

Interface handler of slave node

Definition at line 524 of file lin.h.

**16.90.2.10.18** `lin_tl_pdu_data_t* tl_rx_queue_data_ptr`

Rx queue data

Definition at line 529 of file lin.h.

**16.90.2.10.19** `lin_tl_pdu_data_t* tl_tx_queue_data_ptr`

Tx queue data

Definition at line 528 of file lin.h.

**16.90.2.11** `struct lin_master_data_t`

LIN master configuration structure Implements : `lin_master_data_t_Class`.

Definition at line 538 of file lin.h.

## Data Fields

- `I_u8` [active\\_schedule\\_id](#)
- `I_u8` [previous\\_schedule\\_id](#)
- `I_u8 *` [schedule\\_start\\_entry\\_ptr](#)
- `I_bool` [event\\_trigger\\_collision\\_flg](#)
- `I_u8` [master\\_data\\_buffer](#) [8]
- `I_u16` [frm\\_offset](#)
- `I_u8` [frm\\_size](#)
- `I_u16` [flag\\_offset](#)
- `I_u8` [flag\\_size](#)
- `I_bool` [send\\_slave\\_res\\_flg](#)
- `I_bool` [send\\_functional\\_request\\_flg](#)

## Field Documentation

### 16.90.2.11.1 `I_u8` [active\\_schedule\\_id](#)

Active schedule table id

Definition at line 540 of file `lin.h`.

### 16.90.2.11.2 `I_bool` [event\\_trigger\\_collision\\_flg](#)

Flag trigger collision event

Definition at line 543 of file `lin.h`.

### 16.90.2.11.3 `I_u16` [flag\\_offset](#)

Flag offset

Definition at line 547 of file `lin.h`.

### 16.90.2.11.4 `I_u8` [flag\\_size](#)

Flag size

Definition at line 548 of file `lin.h`.

### 16.90.2.11.5 `I_u16` [frm\\_offset](#)

Frame offset

Definition at line 545 of file `lin.h`.

### 16.90.2.11.6 `I_u8` [frm\\_size](#)

Size of frame

Definition at line 546 of file `lin.h`.

### 16.90.2.11.7 `I_u8` [master\\_data\\_buffer](#)[8]

Master data buffer

Definition at line 544 of file `lin.h`.

### 16.90.2.11.8 `I_u8` [previous\\_schedule\\_id](#)

Previous schedule table id

Definition at line 541 of file `lin.h`.

**16.90.2.11.9** `l_u8* schedule_start_entry_ptr`

Start entry of each schedule table

Definition at line 542 of file lin.h.

**16.90.2.11.10** `l_bool send_functional_request_flg`

Flag send Functional Request

Definition at line 550 of file lin.h.

**16.90.2.11.11** `l_bool send_slave_res_flg`

Flag to send Slave Response Schedule

Definition at line 549 of file lin.h.

**16.90.2.12** `struct lin_protocol_state_t`

LIN protocol status structure Implements : `lin_protocol_state_t_Class`.

Definition at line 557 of file lin.h.

**Data Fields**

- `l_u16 baud_rate`
- `l_u8 * response_buffer_ptr`
- `l_u8 response_length`
- `l_u8 successful_transfer`
- `l_u8 error_in_response`
- `l_bool go_to_sleep_flg`
- `l_u8 current_id`
- `l_u8 last_pid`
- `l_u8 num_of_processed_frame`
- `l_u8 overrun_flg`
- `lin_word_status_str_t word_status`
- `l_u8 next_transmit_tick`
- `l_bool save_config_flg`
- `l_diagnostic_mode_t diagnostic_mode`
- `l_u16 frame_timeout_cnt`
- `l_u16 idle_timeout_cnt`
- `l_bool transmit_error_resp_sig_flg`

**Field Documentation****16.90.2.12.1** `l_u16 baud_rate`

Adjusted baud rate

Definition at line 560 of file lin.h.

**16.90.2.12.2** `l_u8 current_id`

Current PID

Definition at line 566 of file lin.h.

**16.90.2.12.3** `l_diagnostic_mode_t diagnostic_mode`

Diagnostic mode

Definition at line 573 of file lin.h.

**16.90.2.12.4 I\_u8 error\_in\_response**

Error response

Definition at line 564 of file lin.h.

**16.90.2.12.5 I\_u16 frame\_timeout\_cnt**

Frame timeout counter for monitoring if timeout occurs during data transferring

Definition at line 574 of file lin.h.

**16.90.2.12.6 I\_bool go\_to\_sleep\_flg**

Go to sleep flag

Definition at line 565 of file lin.h.

**16.90.2.12.7 I\_u16 idle\_timeout\_cnt**

Idle timeout counter

Definition at line 575 of file lin.h.

**16.90.2.12.8 I\_u8 last\_pid**

Last PID

Definition at line 567 of file lin.h.

**16.90.2.12.9 I\_u8 next\_transmit\_tick**

Used to count the next transmit tick

Definition at line 571 of file lin.h.

**16.90.2.12.10 I\_u8 num\_of\_processed\_frame**

Number of processed frames

Definition at line 568 of file lin.h.

**16.90.2.12.11 I\_u8 overrun\_flg**

overrun flag

Definition at line 569 of file lin.h.

**16.90.2.12.12 I\_u8\* response\_buffer\_ptr**

Response buffer

Definition at line 561 of file lin.h.

**16.90.2.12.13 I\_u8 response\_length**

Response length

Definition at line 562 of file lin.h.

**16.90.2.12.14 I\_bool save\_config\_flg**

Set when save configuration request has been received

Definition at line 572 of file lin.h.



**16.90.2.12.15** `I_u8 successful_transfer`

Transfer flag

Definition at line 563 of file lin.h.

**16.90.2.12.16** `I_bool transmit_error_resp_sig_flg`

Flag indicates that the error response signal is going to be sent

Definition at line 576 of file lin.h.

**16.90.2.12.17** `lin_word_status_str_t word_status`

Word status

Definition at line 570 of file lin.h.

**16.90.3** Macro Definition Documentation**16.90.3.1** `#define CALLBACK_HANDLER( iii, event_id, id ) lin_pid_resp_callback_handler((iii), (event_id), (id))`

`CALLBACK_HANDLER.`

Note

call [lin\\_pid\\_resp\\_callback\\_handler\(\)](#) function in MASTER mode

Definition at line 684 of file lin.h.

**16.90.3.2** `#define LD_NEGATIVE_RESPONSE 0x53U`

Negative response

Definition at line 84 of file lin.h.

**16.90.3.3** `#define LD_POSITIVE_RESPONSE 0x54U`

Positive response

Definition at line 85 of file lin.h.

**16.90.3.4** `#define LIN_LLD_ERROR 0xFFU`

Return value is ERROR

Definition at line 89 of file lin.h.

**16.90.3.5** `#define LIN_LLD_OK 0x00U`

Return value is OK

Definition at line 88 of file lin.h.

**16.90.3.6** `#define LIN_MASTER 1`

Master node

Definition at line 166 of file lin.h.

**16.90.3.7** `#define LIN_READ_USR_DEF_MAX 63U`

Max user defined

Definition at line 81 of file lin.h.

**16.90.3.8 #define LIN\_READ\_USR\_DEF\_MIN 32U**

Min user defined

Definition at line 80 of file lin.h.

**16.90.3.9 #define LIN\_SLAVE 0**

Mode of LIN node (master or slave)

Slave node

Definition at line 165 of file lin.h.

**16.90.3.10 #define LIN\_TL\_CALLBACK\_HANDLER( *iii*, *tl\_event\_id*, *id* ) lin\_tl\_callback\_handler((iii), (tl\_event\_id), (id))**

Definition at line 372 of file lin.h.

**16.90.3.11 #define PCI\_RES\_ASSIGN\_FRAME\_ID\_RANGE 0x01U**

PCI response value assign frame id range

Definition at line 77 of file lin.h.

**16.90.3.12 #define PCI\_RES\_READ\_BY\_IDENTIFY 0x06U**

PCI response value read by identify

Definition at line 75 of file lin.h.

**16.90.3.13 #define PCI\_RES\_SAVE\_CONFIGURATION 0x01U**

PCI response value save configuration

Definition at line 76 of file lin.h.

**16.90.3.14 #define PCI\_SAVE\_CONFIGURATION 0x01U**

PCI value save configuration

Definition at line 72 of file lin.h.

**16.90.3.15 #define SERVICE\_FAULT\_MEMORY\_CLEAR 0x14U**

Service fault memory clear

Definition at line 69 of file lin.h.

**16.90.3.16 #define SERVICE\_ASSIGN\_FRAME\_ID 0xB1U**

Assign frame id service

Definition at line 58 of file lin.h.

**16.90.3.17 #define SERVICE\_ASSIGN\_FRAME\_ID\_RANGE 0xB7U**

Assign frame id range service

Definition at line 62 of file lin.h.

**16.90.3.18 #define SERVICE\_ASSIGN\_NAD 0xB0U**

Assign NAD service

Definition at line 57 of file lin.h.

16.90.3.19 `#define SERVICE_CONDITIONAL_CHANGE_NAD 0xB3U`

Conditional change NAD service

Definition at line 60 of file lin.h.

16.90.3.20 `#define SERVICE_FAULT_MEMORY_READ 0x19U`

Service fault memory read

Definition at line 68 of file lin.h.

16.90.3.21 `#define SERVICE_IO_CONTROL_BY_IDENTIFY 0x2FU`

Service I/O control

Definition at line 67 of file lin.h.

16.90.3.22 `#define SERVICE_READ_BY_IDENTIFY 0xB2U`

Read by identify service

Definition at line 59 of file lin.h.

16.90.3.23 `#define SERVICE_READ_DATA_BY_IDENTIFY 0x22U`

Service read data by identifier

Definition at line 64 of file lin.h.

16.90.3.24 `#define SERVICE_SAVE_CONFIGURATION 0xB6U`

Save configuration service

Definition at line 61 of file lin.h.

16.90.3.25 `#define SERVICE_SESSION_CONTROL 0x10U`

Service session control

Definition at line 66 of file lin.h.

16.90.3.26 `#define SERVICE_WRITE_DATA_BY_IDENTIFY 0x2EU`

Service write data by identifier

Definition at line 65 of file lin.h.

## 16.90.4 Typedef Documentation

16.90.4.1 `typedef I_u8 lin_tl_pdu_data_t[8]`

PDU data. Implements : `lin_tl_pdu_data_t_Class`.

Definition at line 95 of file lin.h.

16.90.4.2 `typedef I_u8 lin_tl_queue_t[8]`

LIN transport layer queue Implements : `lin_tl_queue_t_Class`.

Definition at line 267 of file lin.h.

## 16.90.5 Enumeration Type Documentation

16.90.5.1 enum `diag_interleaved_state_t`

State of diagnostic interleaved mode Implements : `diag_interleaved_state_t_Class`.

## Enumerator

***DIAG\_NOT\_START*** Not into slave response schedule with interleaved mode

***DIAG\_NO\_RESPONSE*** Master send 0x3D but slave does not response

***DIAG\_RESPONSE*** Response receive

Definition at line 450 of file `lin.h`.

16.90.5.2 enum `l_diagnostic_mode_t`

Diagnostic mode Implements : `l_diagnostic_mode_t_Class`.

## Enumerator

***DIAG\_NONE*** None

***DIAG\_INTERLEAVE\_MODE*** Interleave mode

***DIAG\_ONLY\_MODE*** Diagnostic only mode

Definition at line 311 of file `lin.h`.

16.90.5.3 enum `ld_queue_status_t`

Status of queue Implements : `ld_queue_status_t_Class`.

## Enumerator

***LD\_NO\_DATA*** Rx Queue is empty, has no data

***LD\_DATA\_AVAILABLE*** Data in queue is available

***LD\_RECEIVE\_ERROR*** Receive data is error for LIN21 and above

***LD\_QUEUE\_FULL*** The queue is full

***LD\_QUEUE\_AVAILABLE*** Queue is available for insert data for LIN21 and above

***LD\_QUEUE\_EMPTY*** Tx Queue is empty

***LD\_TRANSMIT\_ERROR*** Error while transmitting for LIN21 and above

***LD\_TRANSFER\_ERROR*** Error while transmitting/receiving for LIN20 and J2602

Definition at line 378 of file `lin.h`.

16.90.5.4 enum `lin_diagnostic_class_t`

List of diagnostic classes Implements : `lin_diagnostic_class_t_Class`.

## Enumerator

***LIN\_DIAGNOSTIC\_CLASS\_I*** LIN Diagnostic Class 1

***LIN\_DIAGNOSTIC\_CLASS\_II*** LIN Diagnostic Class 2

***LIN\_DIAGNOSTIC\_CLASS\_III*** LIN Diagnostic Class 3

Definition at line 137 of file `lin.h`.

16.90.5.5 enum `lin_diagnostic_state_t`

LIN diagnostic state Implements : `lin_diagnostic_state_t_Class`.

## Enumerator

**`LD_DIAG_IDLE`** IDLE  
**`LD_DIAG_TX_PHY`** Diagnostic transmit physical  
**`LD_DIAG_TX_FUNCTIONAL`** Diagnostic transmit active  
**`LD_DIAG_TX_INTERLEAVED`** Diagnostic transmit in interleave mode  
**`LD_DIAG_RX_PHY`** Diagnostic receive in physical  
**`LD_DIAG_RX_FUNCTIONAL`** Diagnostic receive functional request  
**`LD_DIAG_RX_INTERLEAVED`** Diagnostic receive in interleave mode

Definition at line 409 of file `lin.h`.

16.90.5.6 enum `lin_frame_response_t`

LIN frame response Implements : `lin_frame_response_t_Class`.

## Enumerator

**`LIN_RES_PUB`** Publisher response  
**`LIN_RES_SUB`** Subscriber response

Definition at line 228 of file `lin.h`.

16.90.5.7 enum `lin_frame_type_t`

Types of frame Implements : `lin_frame_type_t_Class`.

## Enumerator

**`LIN_FRM_UNCD`** Unconditional frame  
**`LIN_FRM_EVNT`** Event triggered frame  
**`LIN_FRM_SPRDC`** Sporadic frame  
**`LIN_FRM_DIAG`** Diagnostic frame

Definition at line 216 of file `lin.h`.

16.90.5.8 enum `lin_last_cfg_result_t`

Status of the last configuration call completed Implements : `lin_last_cfg_result_t_Class`.

## Enumerator

**`LD_SUCCESS`** The service was successfully carried out  
**`LD_NEGATIVE`** The service failed, more information can be found by parsing `error_code`  
**`LD_NO_RESPONSE`** No response was received on the request  
**`LD_OVERWRITTEN`** The slave response frame has been overwritten by another operation

Definition at line 334 of file `lin.h`.

16.90.5.9 enum `lin_llc_event_id_t`

Event id Implements : `lin_llc_event_id_t_Class`.

## Enumerator

**`LIN_LLC_PID_OK`** `LIN_LLC_PID_OK`  
**`LIN_LLC_TX_COMPLETED`** `LIN_LLC_TX_COMPLETED`  
**`LIN_LLC_RX_COMPLETED`** `LIN_LLC_RX_COMPLETED`  
**`LIN_LLC_PID_ERR`** `LIN_LLC_PID_ERR`  
**`LIN_LLC_FRAME_ERR`** `LIN_LLC_FRAME_ERR`  
**`LIN_LLC_CHECKSUM_ERR`** `LIN_LLC_CHECKSUM_ERR`  
**`LIN_LLC_READBACK_ERR`** `LIN_LLC_READBACK_ERR`  
**`LIN_LLC_NODATA_TIMEOUT`** No data timeout or received part of data but not completed  
**`LIN_LLC_BUS_ACTIVITY_TIMEOUT`** `LIN_LLC_BUS_ACTIVITY_TIMEOUT`

Definition at line 105 of file `lin.h`.

16.90.5.10 enum `lin_message_status_t`

Status of LIN message Implements : `lin_message_status_t_Class`.

## Enumerator

**`LD_NO_MSG`** No message  
**`LD_IN_PROGRESS`** In progress  
**`LD_COMPLETED`** Completed  
**`LD_FAILED`** Failed  
**`LD_N_AS_TIMEOUT`** N\_As timeout  
**`LD_N_CR_TIMEOUT`** N\_Cr timeout  
**`LD_WRONG_SN`** Wrong sequence number

Definition at line 394 of file `lin.h`.

16.90.5.11 enum `lin_message_timeout_type_t`

Types of message timeout Implements : `lin_message_timeout_type_t_Class`.

## Enumerator

**`LD_NO_CHECK_TIMEOUT`** No check timeout  
**`LD_CHECK_N_AS_TIMEOUT`** check N\_As timeout  
**`LD_CHECK_N_CR_TIMEOUT`** check N\_Cr timeout

Definition at line 424 of file `lin.h`.

16.90.5.12 enum `lin_protocol_handle_t`

List of protocols Implements : `lin_protocol_handle_t_Class`.

## Enumerator

**`LIN_PROTOCOL_21`** LIN protocol version 2.1, 2.2  
**`LIN_PROTOCOL_J2602`** J2602 protocol  
**`LIN_PROTOCOL_13`** LIN protocol version 1.3  
**`LIN_PROTOCOL_20`** LIN protocol version 2.0

Definition at line 125 of file `lin.h`.

## 16.90.5.13 enum lin\_sch\_tbl\_type\_t

Types of schedule tables Implements : lin\_sch\_tbl\_type\_t\_Class.

## Enumerator

**LIN\_SCH\_TBL\_NULL** Run nothing  
**LIN\_SCH\_TBL\_NORM** Normal schedule table  
**LIN\_SCH\_TBL\_DIAG** Diagnostic schedule table  
**LIN\_SCH\_TBL\_GO\_TO\_SLEEP** Goto sleep schedule table  
**LIN\_SCH\_TBL\_COLL\_RESOLV** Collision resolving schedule table

Definition at line 273 of file lin.h.

## 16.90.5.14 enum lin\_service\_status\_t

Status of the last configuration call for LIN 2.1 Implements : lin\_service\_status\_t\_Class.

## Enumerator

**LD\_SERVICE\_BUSY** Service is ongoing  
**LD\_REQUEST\_FINISHED** The configuration request has been completed  
**LD\_SERVICE\_IDLE** The configuration request/response combination has been completed  
**LD\_SERVICE\_ERROR** The configuration request or response experienced an error

Definition at line 322 of file lin.h.

## 16.90.5.15 enum lin\_tl\_callback\_return\_t

Transport layer event IDs Implements : lin\_tl\_callback\_return\_t\_Class.

## Enumerator

**TL\_ACTION\_NONE** Default return value of call back function  
**TL\_ACTION\_ID\_IGNORE** Ignore this ID

Definition at line 362 of file lin.h.

## 16.90.5.16 enum lin\_tl\_event\_id\_t

Transport layer event IDs Implements : lin\_tl\_event\_id\_t\_Class.

## Enumerator

**TL\_MAKE\_RES\_DATA** Make master request data  
**TL\_SLAVE\_GET\_ACTION** Get slave action  
**TL\_TX\_COMPLETED** Transmit completed  
**TL\_RX\_COMPLETED** Receive completed  
**TL\_ERROR** Transport error  
**TL\_TIMEOUT\_SERVICE** Transmit timeout  
**TL\_HANDLER\_INTERLEAVE\_MODE** Interleave mode  
**TL\_RECEIVE\_MESSAGE** Return data for Id\_receive\_message function

Definition at line 346 of file lin.h.

## 16.90.6 Function Documentation

16.90.6.1 `I_u8 Id_read_by_id_callout ( I_ifc_handle iii, I_u8 id, I_u8 * data )`

16.90.6.2 `static I_u16 lin_calc_max_header_timeout_cnt ( I_u32 baudRate ) [inline],[static]`

Computes maximum header timeout.

$T_{Header\_Maximum} = 1.4 * T_{Header\_Nominal}$ ,  $T_{Header\_Nominal} = 34 * T_{Bit}$ , (13 nominal bits of break; 1 nominal bit of break delimiter; 10 bits for SYNC and 10 bits of PID)  $TIME\_OUT\_UNIT\_US$  is in micro second

### Parameters

<i>in</i>	<i>baudRate</i>	LIN network baud rate
-----------	-----------------	-----------------------

### Returns

maximum timeout for the selected baud rate

Implements : `lin_calc_max_header_timeout_cnt_Activity`

Definition at line 626 of file `lin.h`.

16.90.6.3 `static I_u16 lin_calc_max_res_timeout_cnt ( I_u32 baudRate, I_u8 size ) [inline],[static]`

Computes the maximum response timeout.

$T_{Response\_Maximum} = 1.4 * T_{Response\_Nominal}$ ,  $T_{Response\_Nominal} = 10 * (N_{Data} + 1) * T_{Bit}$

### Parameters

<i>in</i>	<i>baudRate</i>	LIN network baud rate
<i>in</i>	<i>size</i>	frame size in bytes

### Returns

maximum response timeout for the given baud rate and frame size

Implements : `lin_calc_max_res_timeout_cnt_Activity`

Definition at line 642 of file `lin.h`.

16.90.6.4 `void lin_lld_deinit ( I_ifc_handle iii )`

This function disconnect the node from the cluster and free all hardware used.

### Parameters

<i>in</i>	<i>iii</i>	LIN interface that is being handled
-----------	------------	-------------------------------------

### Returns

Zero for success  
void

Definition at line 153 of file `lin.c`.

16.90.6.5 `lin_node_state_t lin_lld_get_state ( I_ifc_handle iii )`

This function gets current state of an interface.



**Parameters**

<i>in</i>	<i>iii</i>	LIN interface that is being handled
-----------	------------	-------------------------------------

**Returns**

current LIN node state

Definition at line 174 of file lin.c.

**16.90.6.6 I\_u8 lin\_lld\_ignore\_response ( I\_ifc\_handle *iii* )**

This function terminates an on-going data transmission/reception.

**Parameters**

<i>in</i>	<i>iii</i>	LIN interface that is being handled
-----------	------------	-------------------------------------

**Returns**

Zero for success  
Non-zero for error

Definition at line 294 of file lin.c.

**16.90.6.7 I\_bool lin\_lld\_init ( I\_ifc\_handle *iii* )**

This function initializes a LIN hardware instance for operation. This function will initialize the run-time state structure to keep track of the on-going transfers, initialize the module to user defined settings and default settings, configure the IRQ state structure and enable the module-level interrupt to the core, and enable the LIN hardware module transmitter and receiver.

**Parameters**

<i>in</i>	<i>iii</i>	LIN interface that is being handled
-----------	------------	-------------------------------------

**Returns**

zero if the initialization was successful and non-zero if failed

Definition at line 88 of file lin.c.

**16.90.6.8 I\_u8 lin\_lld\_int\_disable ( I\_ifc\_handle *iii* )**

Disable the interrupt related to the interface.

**Parameters**

<i>in</i>	<i>iii</i>	LIN interface that is being handled
-----------	------------	-------------------------------------

**Returns**

Zero for success  
Non-zero for error

Definition at line 271 of file lin.c.

**16.90.6.9 I\_u8 lin\_lld\_int\_enable ( I\_ifc\_handle *iii* )**

Enable the interrupt related to the interface.

**Parameters**

<i>in</i>	<i>iii</i>	LIN interface that is being handled
-----------	------------	-------------------------------------

**Returns**

Zero for success  
Non-zero for error

Definition at line 248 of file lin.c.

**16.90.6.10** `I_u8 lin_ild_rx_response ( I_ifc_handle iii, I_u8 response_length )`

This function receives frame data into the LIN\_ild\_response\_buffer[*iii*] buffer.

This function will prepare LIN interface to receive data and then return. Data bytes will be received to the buffer in the interrupt handler of LIN interface. This function returns zero if preparation of receiving data was successful.

**Parameters**

<i>in</i>	<i>iii</i>	LIN interface that is being handled
<i>in</i>	<i>response_length</i>	Length of response

**Returns**

Zero for success  
Non-zero for error

Definition at line 376 of file lin.c.

**16.90.6.11** `I_u8 lin_ild_set_low_power_mode ( I_ifc_handle iii )`

Let the low level driver go to low power mode.

**Parameters**

<i>in</i>	<i>iii</i>	LIN interface that is being handled
-----------	------------	-------------------------------------

**Returns**

Zero for success  
Non-zero for error

Definition at line 317 of file lin.c.

**16.90.6.12** `I_u8 lin_ild_set_response ( I_ifc_handle iii, I_u8 response_length )`

This function sends frame data that is contained in LIN\_ild\_response\_buffer[*iii*].

This function will send the first data byte in the buffer and then return. Next data bytes will be sent in the interrupt handler of LIN interface. This function returns zero if sending of first data byte was successful.

**Parameters**

<i>in</i>	<i>iii</i>	LIN interface that is being handled
<i>in</i>	<i>response_length</i>	Length of response

**Returns**

Zero for success  
Non-zero for error

Definition at line 340 of file lin.c.

**16.90.6.13 void lin\_ild\_timeout\_service ( I\_ifc\_handle *iii* )**

Callback function for Timer Interrupt Handler In timer IRQ handler, call this function. Used to check if frame timeout has occurred during frame data transmission and reception, to check for N\_As and N\_Cr timeout for LIN 2.1 and above. This function also check if there is no LIN bus communication (no headers and no frame data transferring) for Idle timeout (s), then put LIN node to Sleep mode. Users may initialize a timer (for example FTM) with period of Timeout unit (default: 500 micro seconds) to call [lin\\_ild\\_timeout\\_service\(\)](#). For an interface *iii*, Idle timeout (s) = max\_idle\_timeout\_cnt \* Timeout unit (us) frame timeout (us) = frame\_timeout\_cnt \* Timeout unit (us) N\_As timeout (us) = N\_As\_timeout \* Timeout unit (us) N\_Cr timeout (us) = N\_Cr\_timeout \* Timeout unit (us)

**Parameters**

<i>in</i>	<i>iii</i>	LIN interface that is being handled
-----------	------------	-------------------------------------

**Returns**

void

Definition at line 407 of file lin.c.

**16.90.6.14 I\_u8 lin\_ild\_tx\_header ( I\_ifc\_handle *iii*, I\_u8 *id* )**

This function sends frame header for the input PID.

This function only initializes the sending of break field and then return. Then the sync byte and PID will be sent in the interrupt handler of LIN interface.

**Parameters**

<i>in</i>	<i>iii</i>	LIN interface that is being handled
<i>in</i>	<i>id</i>	ID of the header to be sent

**Returns**

Zero for success  
Non-zero for error

Definition at line 197 of file lin.c.

**16.90.6.15 I\_u8 lin\_ild\_tx\_wake\_up ( I\_ifc\_handle *iii* )**

This function send a wakeup signal.

**Parameters**

<i>in</i>	<i>iii</i>	LIN interface that is being handled
-----------	------------	-------------------------------------

**Returns**

Zero for success  
Non-zero for error

Definition at line 225 of file lin.c.

**16.90.6.16 void lin\_pid\_resp\_callback\_handler ( I\_ifc\_handle *iii*, const lin\_ild\_event\_id\_t *event\_id*, I\_u8 *id* )**

Callback handler for low level events.

This callback handler is being called from the LIN driver callback

**Parameters**

in	<i>iii</i>	LIN interface that is being handled
in	<i>event_id</i>	Low level event id <a href="#">lin_llc_event_id_t</a>
in	<i>id</i>	Current protected identifier under processing by driver

Definition at line 74 of file `lin_common_proto.c`.

**16.90.6.17** `I_u8 lin_process_parity ( I_u8 pid, I_u8 typeAction )`

Makes or checks parity bits. If action is checking parity, the function returns ID value if parity bits are correct or 0xFF if parity bits are incorrect. If action is making parity bits, then from input value of ID, the function returns PID.

**Parameters**

<i>pid</i>	PID byte in case of checking parity bits or ID byte in case of making parity bits.
<i>typeAction</i>	TRUE for Checking parity bits, FALSE for making parity bits

**Returns**

0xFF if parity bits are incorrect, ID in case of checking parity bits and they are correct. Function returns PID in case of making parity bits.

Definition at line 71 of file `lin.c`.

**16.90.6.18** `lin_tl_callback_return_t lin_tl_callback_handler ( I_ifc_handle iii, lin_tl_event_id_t tl_event_id, I_u8 id )`

Definition at line 81 of file `lin_common_tl_proto.c`.

**16.90.7 Variable Documentation**

**16.90.7.1** `volatile I_u8 g_buffer_backup_data[8]`

**16.90.7.2** `volatile I_u8 g_lin_flag_handle_tbl[LIN_FLAG_BUF_SIZE]`

**16.90.7.3** `volatile I_u8 g_lin_frame_data_buffer[LIN_FRAME_BUF_SIZE]`

**16.90.7.4** `volatile I_bool g_lin_frame_flag_handle_tbl[LIN_NUM_OF_FRMS]`

**16.90.7.5** `volatile I_u8 g_lin_frame_updating_flag_tbl[LIN_NUM_OF_FRMS]`

**16.90.7.6** `const I_ifc_handle g_lin_hardware_ifc[HARDWARE_INSTANCE_COUNT]`

**16.90.7.7** `lin_master_data_t g_lin_master_data_array[LIN_NUM_OF_MASTER_IFCS]`

Global array for storing the master interfaces configurations

Definition at line 49 of file `lin.c`.

**16.90.7.8** `const lin_node_attribute_t g_lin_node_attribute_array[LIN_NUM_OF_SLAVE_IFCS]`

**16.90.7.9** `lin_protocol_state_t g_lin_protocol_state_array[LIN_NUM_OF_IFCS]`

Global array for storing the protocol state for each interface

Definition at line 47 of file `lin.c`.

**16.90.7.10** `const lin_protocol_user_config_t g_lin_protocol_user_cfg_array[LIN_NUM_OF_IFCS]`

**16.90.7.11** `lin_tl_descriptor_t g_lin_tl_descriptor_array[LIN_NUM_OF_IFCS]`

Global array for storing transport configuration for each interface

Definition at line 46 of file lin.c.

16.90.7.12 `const l_u32 g_lin_virtual_ifc[LIN_NUM_OF_IFCS]`

16.90.7.13 `const lin_timer_get_time_interval_t timerGetTimeIntervalCallbackArr[LIN_NUM_OF_IFCS]`

## 16.91 MPU Driver

### 16.91.1 Detailed Description

Memory Protection Unit Peripheral Driver.

#### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
{S32SDK_PATH}\platform\drivers\src\mpu\mpu_driver.c
{S32SDK_PATH}\platform\drivers\src\mpu\mpu_hw_access.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
{S32SDK_PATH}\platform\drivers\inc\
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

No special dependencies are required for this component

#### Pre-Initialization information of MPU module

1. Before using the MPU driver the protocol clock of the module must be configured by the application using clock module.
2. Bus fault or Hard fault exception must be configured to handle MPU access violation.

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. The default settings for the Region Descriptor 0 (RGD0):

- The access right for **CORE, DMA,..** can be **changed** except **DEBUGGER** master.
- The **start address, end address, process identifier** and **process identifier mask** are **ignored**.

This is example code to configure the MPU driver:

#### 1. Define MPU instance

```
/* MPU 0 */
#define INST_MPU 0U

/* Status variable */
status_t status;
```

#### 2. Configuration

##### User configuration

```
/* Region count */
#define REGION_CNT (1U)

/* Master access configuration
FEATURE_MPU_MASTER_COUNT macro has been already defined (number of masters supported by hardware)
*/
```

```

mpu_master_access_right_t masterAccRight[FEATURE_MPU_MASTER_COUNT] =
{
    /* CORE */
    {
        .masterNum    = FEATURE_MPU_MASTER_CORE,        /* Master number */
        .accessRight = MPU_SUPERVISOR_RWX_USER_RWX,    /* Access right */
        .processIdentifierEnable = false,              /* Process identifier enable */
    },
    /* The rest masters should be defined here */
    ...
}
/* User configuration */
mpu_user_config_t userConfig[REGION_CNT] =
{
    /* Region 0 */
    {
        .startAddr      = 0x00000000U,                /* Memory region start address */
        .endAddr        = 0xFFFFFFFFU,                /* Memory region end address */
        .masterAccRight  = masterAccRight,             /* Master access right */
        .processIdEnable = false,                      /* Process identifier enable */
        .processIdentifier = 0x00U,                    /* Process identifier */
        .processIdMask   = 0x00U                      /* Process identifier mask */
    }
}

```

or get default configuration

```

/* Defines master access right structure */
mpu_master_access_right_t masterAccRight[FEATURE_MPU_MASTER_COUNT];

/* Gets default region configuration
   Cover entire memory
   Access right of all masters are allowed
*/
mpu_user_config_t regionConfig0 =
    MPU_DRV_GetDefaultRegionConfig(masterAccRight);
mpu_user_config_t userConfig[REGION_CNT] =
{
    regionConfig0
};

```

### 3. Initializes

```

/* Initializes the MPU instance */
status = MPU_DRV_Init(INST_MPU, REGION_CNT, userConfig);

```

### 4. De-initializes

```

/* De-initializes the MPU instance */
MPU_DRV_Deinit(INST_MPU);

```

After MPU initialization:

- The [MPU\\_DRV\\_SetRegionConfig\(\)](#) can be used to add/update the new/existing region descriptor.

```

/* Add the new region descriptor.
   Region 1 to be the same with region 0.
*/
status = MPU_DRV_SetRegionConfig(INST_MPU, 1U, &userConfig[0U]);

/* Updates the existing region descriptor.
   Updates PID mask value on region 1.
*/
userConfig[0U].processIdMask = 0xFFU;
status = MPU_DRV_SetRegionConfig(INST_MPU, 1U, &userConfig[0U]);

```

- The [MPU\\_DRV\\_SetRegionAddr\(\)](#) can be used to update the start and end address on an existing region descriptor.

```

/* Updates region 1 location (0x20000000 - 0x2FFFFFFF) */
MPU_DRV_SetRegionAddr(INST_MPU, 1U, 0x20000000U, 0x2FFFFFFFU);

```

- The [MPU\\_DRV\\_SetMasterAccessRights\(\)](#) can be used to update access permission of master in the region.

```

/* DMA can only operate on region 1 */
mpu_master_access_right_t DMA_AccRight;
DMA_AccRight.masterNum = FEATURE_MPU_MASTER_DMA;
/* Removes all access rights of DMA on region 0 */
DMA_AccRight.accessRight = MPU_SUPERVISOR_USER_NONE;
status = MPU_DRV_SetMasterAccessRights(INST_MPU, 0U, &DMA_AccRight);
/* Allows all access to region 1 from DMA */
DMA_AccRight.accessRight = MPU_SUPERVISOR_USER_RWX;
status = MPU_DRV_SetMasterAccessRights(INST_MPU, 1U, &DMA_AccRight);

```

- The `MPU_DRV_GetDetailErrorAccessInfo()` API can be used to get the status of a slave port and the detail when an error occurred.

```

/* DMA access to SRAML
- Access type: read
- Address: 0x1FFEFF00 (region 0)
*/
...
/* Checks and gets error status on slave port 1 (SRAML backdoor) */
bool errStatus = false;
mpu_access_err_info_t errReport;
errStatus = MPU_DRV_GetDetailErrorAccessInfo(INST_MPU,
    FEATURE_MPU_SLAVE_SRAM_BACKDOOR, &errReport);
/* Checks status:
- errStatus: true
- errReport:
- errReport.master: FEATURE_MPU_MASTER_DMA (DMA logical ID)
- errReport.attributes: MPU_DATA_ACCESS_IN_SUPERVISOR_MODE (Data access in supervisor mode)
- errReport.accessType: MPU_ERR_TYPE_READ (Read access)
- errReport.accessCtr: 0x8000 (Access violation occurs on region 0 - MSB is region 0 and so on)
- errReport.addr: 0x1FFEFF00 (Data access in supervisor mode)
- errReport.processorIdentification: 0U (Do not support for non-core processor)
*/
...

```

- The `MPU_DRV_EnableRegion()` can be used to enable or disable region descriptor.

```

/* Disables DMA - disable region 1 descriptor */
MPU_DRV_EnableRegion(INST_MPU, 1U, false);
/* Enables again */
MPU_DRV_EnableRegion(INST_MPU, 1U, true);

```

## Power management:

- To minimizes power dissipation, disables MPU module or regions by using `MPU_DRV_Deinit()/MPU_DRV_↵_EnableRegion()` when they are unused anymore.

## Data Structures

- struct `mpu_access_err_info_t`  
MPU detail error access info Implements : `mpu_access_err_info_t_Class`. [More...](#)
- struct `mpu_master_access_right_t`  
MPU master access rights. Implements : `mpu_master_access_right_t_Class`. [More...](#)
- struct `mpu_user_config_t`  
MPU user region configuration structure. This structure is used when calling the `MPU_DRV_Init` function. Implements : `mpu_user_config_t_Class`. [More...](#)

## Enumerations

- enum `mpu_err_access_type_t` { `MPU_ERR_TYPE_READ` = 0U, `MPU_ERR_TYPE_WRITE` = 1U }  
MPU access error Implements : `mpu_err_access_type_t_Class`.
- enum `mpu_err_attributes_t` { `MPU_INSTRUCTION_ACCESS_IN_USER_MODE` = 0U, `MPU_DATA_ACCESS_IN_USER_MODE` = 1U, `MPU_INSTRUCTION_ACCESS_IN_SUPERVISOR_MODE` = 2U, `MPU_DATA_ACCESS_IN_SUPERVISOR_MODE` = 3U }  
MPU access error attributes Implements : `mpu_err_attributes_t_Class`.



```

enum mpu_access_rights_t {
    MPU_SUPERVISOR_RWX_USER_NONE = 0x00U, MPU_SUPERVISOR_RWX_USER_X = 0x01U, MPU_SUPERVISOR_RWX_USER_W = 0x02U, MPU_SUPERVISOR_RWX_USER_WX = 0x03U,
    MPU_SUPERVISOR_RWX_USER_R = 0x04U, MPU_SUPERVISOR_RWX_USER_RX = 0x05U, MPU_SUPERVISOR_RWX_USER_RW = 0x06U, MPU_SUPERVISOR_RWX_USER_RWX = 0x07U,
    MPU_SUPERVISOR_RX_USER_NONE = 0x08U, MPU_SUPERVISOR_RX_USER_X = 0x09U, MPU_SUPERVISOR_RX_USER_W = 0x0AU, MPU_SUPERVISOR_RX_USER_WX = 0x0BU,
    MPU_SUPERVISOR_RX_USER_R = 0x0CU, MPU_SUPERVISOR_RX_USER_RX = 0x0DU, MPU_SUPERVISOR_RX_USER_RW = 0x0EU, MPU_SUPERVISOR_RX_USER_RWX = 0x0FU,
    MPU_SUPERVISOR_RW_USER_NONE = 0x10U, MPU_SUPERVISOR_RW_USER_X = 0x11U, MPU_SUPERVISOR_RW_USER_W = 0x12U, MPU_SUPERVISOR_RW_USER_WX = 0x13U,
    MPU_SUPERVISOR_RW_USER_R = 0x14U, MPU_SUPERVISOR_RW_USER_RX = 0x15U, MPU_SUPERVISOR_RW_USER_RW = 0x16U, MPU_SUPERVISOR_RW_USER_RWX = 0x17U,
    MPU_SUPERVISOR_USER_NONE = 0x18U, MPU_SUPERVISOR_USER_X = 0x19U, MPU_SUPERVISOR_USER_W = 0x1AU, MPU_SUPERVISOR_USER_WX = 0x1BU,
    MPU_SUPERVISOR_USER_R = 0x1CU, MPU_SUPERVISOR_USER_RX = 0x1DU, MPU_SUPERVISOR_USER_RW = 0x1EU, MPU_SUPERVISOR_USER_RWX = 0x1FU,
    MPU_NONE = 0x80U, MPU_W = 0xA0U, MPU_R = 0xC0U, MPU_RW = 0xE0U }

```

MPU access rights.

Code	Supervisor	User	Description
MPU_SUPERVISOR_RWX_USER_NONE	r w x	- - -	Allow Read, write, execute in supervisor mode; no access in user mode
MPU_SUPERVISOR_RWX_USER_X	r w x	- - x	Allow Read, write, execute in supervisor mode; execute in user mode
MPU_SUPERVISOR_RWX_USER_W	r w x	- w -	Allow Read, write, execute in supervisor mode; write in user mode
MPU_SUPERVISOR_RWX_USER_WX	r w x	- w x	Allow Read, write, execute in supervisor mode; write and execute in user mode
MPU_SUPERVISOR_RWX_USER_R	r w x	r - -	Allow Read, write, execute in supervisor mode; read in user mode
MPU_SUPERVISOR_RWX_USER_RX	r w x	r - x	Allow Read, write, execute in supervisor mode; read and execute in user mode
MPU_SUPERVISOR_RWX_USER_RW	r w x	r w -	Allow Read, write, execute in supervisor mode; read and write in user mode
MPU_SUPERVISOR_RWX_USER_RWX	r w x	r w x	Allow Read, write, execute in supervisor mode; read, write and execute in user mode
MPU_SUPERVISOR_RX_USER_NONE	r - x	- - -	Allow Read, execute in supervisor mode; no access in user mode
MPU_SUPERVISOR_RX_USER_X	r - x	- - x	Allow Read, execute in supervisor mode; execute in user mode

MPU_SUPERVISOR_↔ RX_USER_W	r - x	- w -	Allow Read, execute in supervisor mode; write in user mode
MPU_SUPERVISOR_↔ RX_USER_WX	r - x	- w x	Allow Read, execute in supervisor mode; write and execute in user mode
MPU_SUPERVISOR_↔ RX_USER_R	r - x	r - -	Allow Read, execute in supervisor mode; read in user mode
MPU_SUPERVISOR_↔ RX_USER_RX	r - x	r - x	Allow Read, execute in supervisor mode; read and execute in user mode
MPU_SUPERVISOR_↔ RX_USER_RW	r - x	r w -	Allow Read, execute in supervisor mode; read and write in user mode
MPU_SUPERVISOR_↔ RX_USER_RWX	r - x	r w x	Allow Read, execute in supervisor mode; read, write and execute in user mode
MPU_SUPERVISOR_↔ RW_USER_NONE	r w -	- - -	Allow Read, write in supervisor mode; no access in user mode
MPU_SUPERVISOR_↔ RW_USER_X	r w -	- - x	Allow Read, write in supervisor mode; execute in user mode
MPU_SUPERVISOR_↔ RW_USER_W	r w -	- w -	Allow Read, write in supervisor mode; write in user mode
MPU_SUPERVISOR_↔ RW_USER_WX	r w -	- w x	Allow Read, write in supervisor mode; write and execute in user mode
MPU_SUPERVISOR_↔ RW_USER_R	r w -	r - -	Allow Read, write in supervisor mode; read in user mode
MPU_SUPERVISOR_↔ RW_USER_RX	r w -	r - x	Allow Read, write in supervisor mode; read and execute in user mode
MPU_SUPERVISOR_↔ RW_USER_RW	r w -	r w -	Allow Read, write in supervisor mode; read and write in user mode
MPU_SUPERVISOR_↔ RW_USER_RWX	r w -	r w x	Allow Read, write in supervisor mode; read, write and execute in user mode
MPU_SUPERVISOR_↔ USER_NONE	- - -	- - -	No access allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_X	- - x	- - x	Execute operation is allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_W	- w -	- w -	Write operation is allowed in user and supervisor modes

<code>MPU_SUPERVISOR_↔ USER_WX</code>	<code>- w x</code>	<code>- w x</code>	Write and execute operations are allowed in user and supervisor modes
<code>MPU_SUPERVISOR_↔ USER_R</code>	<code>r - -</code>	<code>r - -</code>	Read operation is allowed in user and supervisor modes
<code>MPU_SUPERVISOR_↔ USER_RX</code>	<code>r - x</code>	<code>r - x</code>	Read and execute operations are allowed in user and supervisor modes
<code>MPU_SUPERVISOR_↔ USER_RW</code>	<code>r w -</code>	<code>r w -</code>	Read and write operations are allowed in user and supervisor modes
<code>MPU_SUPERVISOR_↔ USER_RWX</code>	<code>r w x</code>	<code>r w x</code>	Read write and execute operations are allowed in user and supervisor modes

## MPU Driver API

- `status_t MPU_DRV_Init` (uint32\_t instance, uint8\_t regionCnt, const `mpu_user_config_t` \*userConfigArr)  
*The function initializes the memory protection unit by setting the access configurations of all available masters, process identifier and the memory location for the given regions; and activate module finally. Please note that access rights for region 0 will always be configured and regionCnt takes values between 1 and the maximum region count supported by the hardware. e.g. In S32K144 the number of supported regions is 8. The user must make sure that the clock is enabled.*
- `void MPU_DRV_Deinit` (uint32\_t instance)  
*De-initializes the memory protection unit by resetting all regions to default and disable module.*
- `void MPU_DRV_SetRegionAddr` (uint32\_t instance, uint8\_t regionNum, uint32\_t startAddr, uint32\_t endAddr)  
*Sets the region start and end address.*
- `status_t MPU_DRV_SetRegionConfig` (uint32\_t instance, uint8\_t regionNum, const `mpu_user_config_↔  
t` \*userConfigPtr)  
*Sets the region configuration. Updates the access configuration of all available masters, process identifier and memory location in a given region.*
- `status_t MPU_DRV_SetMasterAccessRights` (uint32\_t instance, uint8\_t regionNum, const `mpu_master_↔  
access_right_t` \*accessRightsPtr)  
*Configures access permission for bus master in region.*
- `bool MPU_DRV_GetDetailErrorAccessInfo` (uint32\_t instance, uint8\_t slavePortNum, `mpu_access_err_info_↔  
_t` \*errInfoPtr)  
*Checks and gets the MPU access error detail information for a slave port. Clears bus error flag if an error occurs.*
- `mpu_user_config_t MPU_DRV_GetDefaultRegionConfig` (`mpu_master_access_right_t` \*masterAccRight)  
*Gets default region configuration. Grants all access rights for masters and disables PID on entire memory.*
- `void MPU_DRV_EnableRegion` (uint32\_t instance, uint8\_t regionNum, bool enable)  
*Enables/Disables region descriptor. Please note that region 0 should not be disabled.*

## 16.91.2 Data Structure Documentation

### 16.91.2.1 struct mpu\_access\_err\_info\_t

MPU detail error access info Implements : `mpu_access_err_info_t`\_Class.

Definition at line 63 of file `mpu_driver.h`.

**Data Fields**

- [uint8\\_t master](#)
- [mpu\\_err\\_attributes\\_t attributes](#)
- [mpu\\_err\\_access\\_type\\_t accessType](#)
- [uint16\\_t accessCtr](#)
- [uint32\\_t addr](#)

**Field Documentation****16.91.2.1.1 uint16\_t accessCtr**

Access error control

Definition at line 68 of file mpu\_driver.h.

**16.91.2.1.2 mpu\_err\_access\_type\_t accessType**

Access error type

Definition at line 67 of file mpu\_driver.h.

**16.91.2.1.3 uint32\_t addr**

Access error address

Definition at line 69 of file mpu\_driver.h.

**16.91.2.1.4 mpu\_err\_attributes\_t attributes**

Access error attributes

Definition at line 66 of file mpu\_driver.h.

**16.91.2.1.5 uint8\_t master**

Access error master

Definition at line 65 of file mpu\_driver.h.

**16.91.2.2 struct mpu\_master\_access\_right\_t**

MPU master access rights. Implements : mpu\_master\_access\_right\_t\_Class.

Definition at line 173 of file mpu\_driver.h.

**Data Fields**

- [uint8\\_t masterNum](#)
- [mpu\\_access\\_rights\\_t accessRight](#)

**Field Documentation****16.91.2.2.1 mpu\_access\_rights\_t accessRight**

Access right

Definition at line 176 of file mpu\_driver.h.

**16.91.2.2.2 uint8\_t masterNum**

Master number

Definition at line 175 of file mpu\_driver.h.

## 16.91.2.3 struct mpu\_user\_config\_t

MPU user region configuration structure. This structure is used when calling the MPU\_DRV\_Init function. Implements : mpu\_user\_config\_t\_Class.

Definition at line 187 of file mpu\_driver.h.

## Data Fields

- uint32\_t startAddr
- uint32\_t endAddr
- const mpu\_master\_access\_right\_t \* masterAccRight

## Field Documentation

## 16.91.2.3.1 uint32\_t endAddr

Memory region end address

Definition at line 190 of file mpu\_driver.h.

## 16.91.2.3.2 const mpu\_master\_access\_right\_t \* masterAccRight

Access permission for masters

Definition at line 191 of file mpu\_driver.h.

## 16.91.2.3.3 uint32\_t startAddr

Memory region start address

Definition at line 189 of file mpu\_driver.h.

## 16.91.3 Enumeration Type Documentation

## 16.91.3.1 enum mpu\_access\_rights\_t

MPU access rights.

Code	Supervisor	User	Description
MPU_SUPERVISOR_↔ RWX_USER_NONE	r w x	- - -	Allow Read, write, execute in supervisor mode; no access in user mode
MPU_SUPERVISOR_↔ RWX_USER_X	r w x	- - x	Allow Read, write, execute in supervisor mode; execute in user mode
MPU_SUPERVISOR_↔ RWX_USER_W	r w x	- w -	Allow Read, write, execute in supervisor mode; write in user mode
MPU_SUPERVISOR_↔ RWX_USER_WX	r w x	- w x	Allow Read, write, execute in supervisor mode; write and execute in user mode

MPU_SUPERVISOR_↔ RWX_USER_R	r w x	r - -	Allow Read, write, execute in supervisor mode; read in user mode
MPU_SUPERVISOR_↔ RWX_USER_RX	r w x	r - x	Allow Read, write, execute in supervisor mode; read and execute in user mode
MPU_SUPERVISOR_↔ RWX_USER_RW	r w x	r w -	Allow Read, write, execute in supervisor mode; read and write in user mode
MPU_SUPERVISOR_↔ RWX_USER_RWX	r w x	r w x	Allow Read, write, execute in supervisor mode; read, write and execute in user mode
MPU_SUPERVISOR_↔ RX_USER_NONE	r - x	- - -	Allow Read, execute in supervisor mode; no access in user mode
MPU_SUPERVISOR_↔ RX_USER_X	r - x	- - x	Allow Read, execute in supervisor mode; execute in user mode
MPU_SUPERVISOR_↔ RX_USER_W	r - x	- w -	Allow Read, execute in supervisor mode; write in user mode
MPU_SUPERVISOR_↔ RX_USER_WX	r - x	- w x	Allow Read, execute in supervisor mode; write and execute in user mode
MPU_SUPERVISOR_↔ RX_USER_R	r - x	r - -	Allow Read, execute in supervisor mode; read in user mode
MPU_SUPERVISOR_↔ RX_USER_RX	r - x	r - x	Allow Read, execute in supervisor mode; read and execute in user mode
MPU_SUPERVISOR_↔ RX_USER_RW	r - x	r w -	Allow Read, execute in supervisor mode; read and write in user mode
MPU_SUPERVISOR_↔ RX_USER_RWX	r - x	r w x	Allow Read, execute in supervisor mode; read, write and execute in user mode
MPU_SUPERVISOR_↔ RW_USER_NONE	r w -	- - -	Allow Read, write in supervisor mode; no access in user mode
MPU_SUPERVISOR_↔ RW_USER_X	r w -	- - x	Allow Read, write in supervisor mode; execute in user mode
MPU_SUPERVISOR_↔ RW_USER_W	r w -	- w -	Allow Read, write in supervisor mode; write in user mode

MPU_SUPERVISOR_↔ RW_USER_WX	r w -	- w x	Allow Read, write in supervisor mode; write and execute in user mode
MPU_SUPERVISOR_↔ RW_USER_R	r w -	r - -	Allow Read, write in supervisor mode; read in user mode
MPU_SUPERVISOR_↔ RW_USER_RX	r w -	r - x	Allow Read, write in supervisor mode; read and execute in user mode
MPU_SUPERVISOR_↔ RW_USER_RW	r w -	r w -	Allow Read, write in supervisor mode; read and write in user mode
MPU_SUPERVISOR_↔ RW_USER_RWX	r w -	r w x	Allow Read, write in supervisor mode; read, write and execute in user mode
MPU_SUPERVISOR_↔ USER_NONE	- - -	- - -	No access allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_X	- - x	- - x	Execute operation is allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_W	- w -	- w -	Write operation is allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_WX	- w x	- w x	Write and execute operations are allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_R	r - -	r - -	Read operation is allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_RX	r - x	r - x	Read and execute operations are allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_RW	r w -	r w -	Read and write operations are allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_RWX	r w x	r w x	Read write and execute operations are allowed in user and supervisor modes

Code	Read/Write permission	Description
MPU_NONE	- -	No Read/Write access permission
MPU_W	- w	Write access permission
MPU_R	r -	Read access permission
MPU_RW	r w	Read/Write access permission

Implements : mpu\_access\_rights\_t\_Class

#### Enumerator

**MPU\_SUPERVISOR\_RWX\_USER\_NONE** 0b00000000U : rwx|—

**MPU\_SUPERVISOR\_RWX\_USER\_X** 0b00000001U : rwx|—x

**MPU\_SUPERVISOR\_RWX\_USER\_W** 0b00000010U : rwx|—w-

```

MPU_SUPERVISOR_RWX_USER_WX 0b00000011U : rwx|~wx
MPU_SUPERVISOR_RWX_USER_R 0b00000100U : rwx|r~
MPU_SUPERVISOR_RWX_USER_RX 0b00000101U : rwx|r~x
MPU_SUPERVISOR_RWX_USER_RW 0b00000110U : rwx|rw~
MPU_SUPERVISOR_RWX_USER_RWX 0b00000111U : rwx|rwx
MPU_SUPERVISOR_RX_USER_NONE 0b00001000U : r~x|~
MPU_SUPERVISOR_RX_USER_X 0b00001001U : r~x|~x
MPU_SUPERVISOR_RX_USER_W 0b00001010U : r~x|~w~
MPU_SUPERVISOR_RX_USER_WX 0b00001011U : r~x|~wx
MPU_SUPERVISOR_RX_USER_R 0b00001100U : r~x|r~
MPU_SUPERVISOR_RX_USER_RX 0b00001101U : r~x|r~x
MPU_SUPERVISOR_RX_USER_RW 0b00001110U : r~x|rw~
MPU_SUPERVISOR_RX_USER_RWX 0b00001111U : r~x|rwx
MPU_SUPERVISOR_RW_USER_NONE 0b00010000U : rw~|~
MPU_SUPERVISOR_RW_USER_X 0b00010001U : rw~|~x
MPU_SUPERVISOR_RW_USER_W 0b00010010U : rw~|~w~
MPU_SUPERVISOR_RW_USER_WX 0b00010011U : rw~|~wx
MPU_SUPERVISOR_RW_USER_R 0b00010100U : rw~|r~
MPU_SUPERVISOR_RW_USER_RX 0b00010101U : rw~|r~x
MPU_SUPERVISOR_RW_USER_RW 0b00010110U : rw~|rw~
MPU_SUPERVISOR_RW_USER_RWX 0b00010111U : rw~|rwx
MPU_SUPERVISOR_USER_NONE 0b00011000U : ~|~
MPU_SUPERVISOR_USER_X 0b00011001U : ~x|~x
MPU_SUPERVISOR_USER_W 0b00011010U : ~w~|~w~
MPU_SUPERVISOR_USER_WX 0b00011011U : ~wx|~wx
MPU_SUPERVISOR_USER_R 0b00011100U : r~|r~
MPU_SUPERVISOR_USER_RX 0b00011101U : r~x|r~x
MPU_SUPERVISOR_USER_RW 0b00011110U : rw~|rw~
MPU_SUPERVISOR_USER_RWX 0b00011111U : rwx|rwx
MPU_NONE 0b10000000U : ~
MPU_W 0b10100000U : w~
MPU_R 0b11000000U : ~r
MPU_RW 0b11100000U : wr

```

Definition at line 121 of file mpu\_driver.h.

#### 16.91.3.2 enum mpu\_err\_access\_type\_t

MPU access error Implements : mpu\_err\_access\_type\_t\_Class.

##### Enumerator

```

MPU_ERR_TYPE_READ MPU error type: read
MPU_ERR_TYPE_WRITE MPU error type: write

```

Definition at line 41 of file mpu\_driver.h.



## 16.91.3.3 enum mpu\_err\_attributes\_t

MPU access error attributes Implements : mpu\_err\_attributes\_t\_Class.

## Enumerator

**MPU\_INSTRUCTION\_ACCESS\_IN\_USER\_MODE** Access instruction error in user mode

**MPU\_DATA\_ACCESS\_IN\_USER\_MODE** Access data error in user mode

**MPU\_INSTRUCTION\_ACCESS\_IN\_SUPERVISOR\_MODE** Access instruction error in supervisor mode

**MPU\_DATA\_ACCESS\_IN\_SUPERVISOR\_MODE** Access data error in supervisor mode

Definition at line 51 of file mpu\_driver.h.

## 16.91.4 Function Documentation

## 16.91.4.1 void MPU\_DRV\_Deinit ( uint32\_t instance )

De-initializes the memory protection unit by resetting all regions to default and disable module.

## Parameters

in	instance	The MPU peripheral instance number.
----	----------	-------------------------------------

Definition at line 105 of file mpu\_driver.c.

## 16.91.4.2 void MPU\_DRV\_EnableRegion ( uint32\_t instance, uint8\_t regionNum, bool enable )

Enables/Disables region descriptor. Please note that region 0 should not be disabled.

## Parameters

in	instance	The MPU peripheral instance number.
in	regionNum	The region number.
in	enable	Valid state <ul style="list-style-type: none"> <li>• true : Enable region.</li> <li>• false : Disable region.</li> </ul>

Definition at line 344 of file mpu\_driver.c.

## 16.91.4.3 mpu\_user\_config\_t MPU\_DRV\_GetDefaultRegionConfig ( mpu\_master\_access\_right\_t \* masterAccRight )

Gets default region configuration. Grants all access rights for masters and disables PID on entire memory.

## Parameters

out	masterAccRight	The pointer to master configuration structure, see <a href="#">mpu_master_access_right_t</a> . The length of array should be defined by number of masters supported by hardware.
-----	----------------	--

## Returns

The default region configuration, see [mpu\\_user\\_config\\_t](#).

Definition at line 308 of file mpu\_driver.c.

## 16.91.4.4 bool MPU\_DRV\_GetDetailErrorAccessInfo ( uint32\_t instance, uint8\_t slavePortNum, mpu\_access\_err\_info\_t \* errInfoPtr )

Checks and gets the MPU access error detail information for a slave port. Clears bus error flag if an error occurs.

**Parameters**

in	<i>instance</i>	The MPU peripheral instance number.
in	<i>slavePortNum</i>	The slave port number to get Error Detail.
out	<i>errInfoPtr</i>	The pointer to access error info structure, #see <a href="#">mpu_access_err_info_t</a> .

**Returns**

operation status

- true : An error has occurred.
- false : No error has occurred.

Definition at line 256 of file mpu\_driver.c.

**16.91.4.5** `status_t MPU_DRV_Init ( uint32_t instance, uint8_t regionCnt, const mpu_user_config_t * userConfigArr )`

The function initializes the memory protection unit by setting the access configurations of all available masters, process identifier and the memory location for the given regions; and activate module finally. Please note that access rights for region 0 will always be configured and regionCnt takes values between 1 and the maximum region count supported by the hardware. e.g. In S32K144 the number of supported regions is 8. The user must make sure that the clock is enabled.

**Parameters**

in	<i>instance</i>	The MPU peripheral instance number.
in	<i>regionCnt</i>	The number of configured regions.
in	<i>userConfigArr</i>	The pointer to the array of MPU user configure structure, see <a href="#">mpu_user_config_t</a> .

**Returns**

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failed due to master number is out of range supported by hardware.

Definition at line 62 of file mpu\_driver.c.

**16.91.4.6** `status_t MPU_DRV_SetMasterAccessRights ( uint32_t instance, uint8_t regionNum, const mpu_master_access_right_t * accessRightsPtr )`

Configures access permission for bus master in region.

**Parameters**

in	<i>instance</i>	The MPU peripheral instance number.
in	<i>regionNum</i>	The MPU region number.
in	<i>accessRightsPtr</i>	The pointer to access permission structure, #see <a href="#">mpu_master_access_right_t</a> .

**Returns**

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failed due to master number is out of range supported by hardware.

Definition at line 224 of file mpu\_driver.c.

**16.91.4.7** `void MPU_DRV_SetRegionAddr ( uint32_t instance, uint8_t regionNum, uint32_t startAddr, uint32_t endAddr )`

Sets the region start and end address.

**Parameters**

in	<i>instance</i>	The MPU peripheral instance number.
in	<i>regionNum</i>	The region number.
in	<i>startAddr</i>	The region start address.
in	<i>endAddr</i>	The region end address.

Definition at line 137 of file mpu\_driver.c.

**16.91.4.8** `status_t MPU_DRV_SetRegionConfig ( uint32_t instance, uint8_t regionNum, const mpu_user_config_t * userConfigPtr )`

Sets the region configuration. Updates the access configuration of all available masters, process identifier and memory location in a given region.

**Parameters**

in	<i>instance</i>	The MPU peripheral instance number.
in	<i>regionNum</i>	The region number.
in	<i>userConfigPtr</i>	The region configuration structure pointer.

**Returns**

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failed due to master number is out of range supported by hardware.

Definition at line 164 of file mpu\_driver.c.

## 16.92 MPU PAL

### 16.92.1 Detailed Description

Memory Protection Unit Peripheral Abstraction Layer.

#### Data Structures

- struct [mpu\\_error\\_info\\_t](#)  
MPU detail error access info Implements : [mpu\\_error\\_info\\_t\\_Class](#). [More...](#)
- struct [mpu\\_master\\_access\\_permission\\_t](#)  
MPU master access permission. Implements : [mpu\\_master\\_access\\_permission\\_t\\_Class](#). [More...](#)
- struct [mpu\\_region\\_config\\_t](#)  
MPU region configuration structure. Implements : [mpu\\_region\\_config\\_t\\_Class](#). [More...](#)

#### Typedefs

- typedef [mpu\\_access\\_rights\\_t](#) [mpu\\_access\\_permission\\_t](#)

MPU detail access permission For specific master:

Code	Supervisor	User	Description
<a href="#">MPU_SUPERVISOR_↔</a> <a href="#">RWX_USER_NONE</a>	<i>r w x</i>	- - -	Allow Read, write, execute in supervisor mode; no access in user mode
<a href="#">MPU_SUPERVISOR_↔</a> <a href="#">RWX_USER_X</a>	<i>r w x</i>	- - x	Allow Read, write, execute in supervisor mode; execute in user mode
<a href="#">MPU_SUPERVISOR_↔</a> <a href="#">RWX_USER_W</a>	<i>r w x</i>	- w -	Allow Read, write, execute in supervisor mode; write in user mode
<a href="#">MPU_SUPERVISOR_↔</a> <a href="#">RWX_USER_WX</a>	<i>r w x</i>	- w x	Allow Read, write, execute in supervisor mode; write and execute in user mode
<a href="#">MPU_SUPERVISOR_↔</a> <a href="#">RWX_USER_R</a>	<i>r w x</i>	r - -	Allow Read, write, execute in supervisor mode; read in user mode
<a href="#">MPU_SUPERVISOR_↔</a> <a href="#">RWX_USER_RX</a>	<i>r w x</i>	r - x	Allow Read, write, execute in supervisor mode; read and execute in user mode
<a href="#">MPU_SUPERVISOR_↔</a> <a href="#">RWX_USER_RW</a>	<i>r w x</i>	r w -	Allow Read, write, execute in supervisor mode; read and write in user mode
<a href="#">MPU_SUPERVISOR_↔</a> <a href="#">RWX_USER_RWX</a>	<i>r w x</i>	r w x	Allow Read, write, execute in supervisor mode; read, write and execute in user mode

<i>MPU_SUPERVISOR_↔ RX_USER_NONE</i>	<i>r - x</i>	<i>- - -</i>	<i>Allow Read, execute in supervisor mode; no access in user mode</i>
<i>MPU_SUPERVISOR_↔ RX_USER_X</i>	<i>r - x</i>	<i>- - x</i>	<i>Allow Read, execute in supervisor mode; execute in user mode</i>
<i>MPU_SUPERVISOR_↔ RX_USER_W</i>	<i>r - x</i>	<i>- w -</i>	<i>Allow Read, execute in supervisor mode; write in user mode</i>
<i>MPU_SUPERVISOR_↔ RX_USER_WX</i>	<i>r - x</i>	<i>- w x</i>	<i>Allow Read, execute in supervisor mode; write and execute in user mode</i>
<i>MPU_SUPERVISOR_↔ RX_USER_R</i>	<i>r - x</i>	<i>r - -</i>	<i>Allow Read, execute in supervisor mode; read in user mode</i>
<i>MPU_SUPERVISOR_↔ RX_USER_RX</i>	<i>r - x</i>	<i>r - x</i>	<i>Allow Read, execute in supervisor mode; read and execute in user mode</i>
<i>MPU_SUPERVISOR_↔ RX_USER_RW</i>	<i>r - x</i>	<i>r w -</i>	<i>Allow Read, execute in supervisor mode; read and write in user mode</i>
<i>MPU_SUPERVISOR_↔ RX_USER_RWX</i>	<i>r - x</i>	<i>r w x</i>	<i>Allow Read, execute in supervisor mode; read, write and execute in user mode</i>
<i>MPU_SUPERVISOR_↔ RW_USER_NONE</i>	<i>r w -</i>	<i>- - -</i>	<i>Allow Read, write in supervisor mode; no access in user mode</i>
<i>MPU_SUPERVISOR_↔ RW_USER_X</i>	<i>r w -</i>	<i>- - x</i>	<i>Allow Read, write in supervisor mode; execute in user mode</i>
<i>MPU_SUPERVISOR_↔ RW_USER_W</i>	<i>r w -</i>	<i>- w -</i>	<i>Allow Read, write in supervisor mode; write in user mode</i>
<i>MPU_SUPERVISOR_↔ RW_USER_WX</i>	<i>r w -</i>	<i>- w x</i>	<i>Allow Read, write in supervisor mode; write and execute in user mode</i>
<i>MPU_SUPERVISOR_↔ RW_USER_R</i>	<i>r w -</i>	<i>r - -</i>	<i>Allow Read, write in supervisor mode; read in user mode</i>
<i>MPU_SUPERVISOR_↔ RW_USER_RX</i>	<i>r w -</i>	<i>r - x</i>	<i>Allow Read, write in supervisor mode; read and execute in user mode</i>
<i>MPU_SUPERVISOR_↔ RW_USER_RW</i>	<i>r w -</i>	<i>r w -</i>	<i>Allow Read, write in supervisor mode; read and write in user mode</i>
<i>MPU_SUPERVISOR_↔ RW_USER_RWX</i>	<i>r w -</i>	<i>r w x</i>	<i>Allow Read, write in supervisor mode; read, write and execute in user mode</i>

<code>MPU_SUPERVISOR_↔ USER_NONE</code>	- - -	- - -	No access allowed in user and supervisor modes
<code>MPU_SUPERVISOR_↔ USER_X</code>	- - x	- - x	Execute operation is allowed in user and supervisor modes
<code>MPU_SUPERVISOR_↔ USER_W</code>	- w -	- w -	Write operation is allowed in user and supervisor modes
<code>MPU_SUPERVISOR_↔ USER_WX</code>	- w x	- w x	Write and execute operations are allowed in user and supervisor modes
<code>MPU_SUPERVISOR_↔ USER_R</code>	r - -	r - -	Read operation is allowed in user and supervisor modes
<code>MPU_SUPERVISOR_↔ USER_RX</code>	r - x	r - x	Read and execute operations are allowed in user and supervisor modes
<code>MPU_SUPERVISOR_↔ USER_RW</code>	r w -	r w -	Read and write operations are allowed in user and supervisor modes
<code>MPU_SUPERVISOR_↔ USER_RWX</code>	r w x	r w x	Read write and execute operations are allowed in user and supervisor modes

## Enumerations

- enum `mpu_error_access_type_t` { `MPU_ERROR_TYPE_READ` = 0U, `MPU_ERROR_TYPE_WRITE` = 1U }  
*MPU access error Implements : `mpu_error_access_type_t` Class.*
- enum `mpu_error_attributes_t` { `MPU_ERROR_USER_MODE_INSTRUCTION_ACCESS` = 0U, `MPU_ERROR_USER_MODE_DATA_ACCESS` = 1U, `MPU_ERROR_SUPERVISOR_MODE_INSTRUCTION_ACCESS` = 2U, `MPU_ERROR_SUPERVISOR_MODE_DATA_ACCESS` = 3U }  
*MPU access error attributes Implements : `mpu_error_attributes_t` Class.*

## MPU PAL API

- status\_t `MPU_Init` (const `mpu_instance_t` \*const instance, uint8\_t regionCnt, const `mpu_region_config_t` \*configPtr)  
*Initializes memory protection unit by allocating regions and granting access rights for masters.*
- status\_t `MPU_Deinit` (const `mpu_instance_t` \*const instance)  
*De-initializes memory protection unit by resetting all regions and masters to default and disable module.*
- status\_t `MPU_GetDefaultRegionConfig` (const `mpu_instance_t` \*const instance, `mpu_master_access_permission_t` \*masterAccRight, `mpu_region_config_t` \*regionConfig)  
*Gets default region configuration. Grants all access rights for masters; disable PID and cache; unlock region descriptor.*
- status\_t `MPU_UpdateRegion` (const `mpu_instance_t` \*const instance, uint8\_t regionNum, const `mpu_region_config_t` \*configPtr)  
*Updates region configuration.*
- status\_t `MPU_EnableRegion` (const `mpu_instance_t` \*const instance, uint8\_t regionNum, bool enable)  
*Enables or disables an exist region configuration.*
- bool `MPU_GetError` (const `mpu_instance_t` \*const instance, uint8\_t channel, `mpu_error_info_t` \*errPtr)  
*Checks and gets the access error detail information then clear error flag if the error caused by a master.*

- enum [mpu\\_inst\\_type\\_t](#)

*Enumeration with the types of peripherals supported by MPU PAL.*

## 16.92.2 Data Structure Documentation

### 16.92.2.1 struct mpu\_error\_info\_t

MPU detail error access info Implements : [mpu\\_error\\_info\\_t\\_Class](#).

Definition at line 67 of file [mpu\\_pal.h](#).

#### Data Fields

- [uint8\\_t](#) [master](#)
- [bool](#) [overrun](#)
- [mpu\\_error\\_attributes\\_t](#) [attributes](#)
- [mpu\\_error\\_access\\_type\\_t](#) [accessType](#)
- [uint32\\_t](#) [accessCtr](#)
- [uint32\\_t](#) [addr](#)
- [uint8\\_t](#) [processId](#)

#### Field Documentation

##### 16.92.2.1.1 [uint32\\_t](#) [accessCtr](#)

Access error control

Definition at line 73 of file [mpu\\_pal.h](#).

##### 16.92.2.1.2 [mpu\\_error\\_access\\_type\\_t](#) [accessType](#)

Access error type

Definition at line 72 of file [mpu\\_pal.h](#).

##### 16.92.2.1.3 [uint32\\_t](#) [addr](#)

Access error address

Definition at line 74 of file [mpu\\_pal.h](#).

##### 16.92.2.1.4 [mpu\\_error\\_attributes\\_t](#) [attributes](#)

Access error attributes

Definition at line 71 of file [mpu\\_pal.h](#).

##### 16.92.2.1.5 [uint8\\_t](#) [master](#)

Access error master

Definition at line 69 of file [mpu\\_pal.h](#).

##### 16.92.2.1.6 [bool](#) [overrun](#)

Access error master overrun

Definition at line 70 of file [mpu\\_pal.h](#).

##### 16.92.2.1.7 [uint8\\_t](#) [processId](#)

Access error process identification

Definition at line 75 of file [mpu\\_pal.h](#).

**16.92.2.2 struct mpu\_master\_access\_permission\_t**

MPU master access permission. Implements : mpu\_master\_access\_permission\_t\_Class.

Definition at line 141 of file mpu\_pal.h.

**Data Fields**

- uint8\_t [masterNum](#)
- [mpu\\_access\\_permission\\_t](#) [accessRight](#)

**Field Documentation****16.92.2.2.1 mpu\_access\_permission\_t accessRight**

Privilege right

Definition at line 144 of file mpu\_pal.h.

**16.92.2.2.2 uint8\_t masterNum**

Master number

Definition at line 143 of file mpu\_pal.h.

**16.92.2.3 struct mpu\_region\_config\_t**

MPU region configuration structure. Implements : mpu\_region\_config\_t\_Class.

Definition at line 151 of file mpu\_pal.h.

**Data Fields**

- uint32\_t [startAddr](#)
- uint32\_t [endAddr](#)
- const [mpu\\_master\\_access\\_permission\\_t](#) \* [masterAccRight](#)
- uint8\_t [processIdEnable](#)
- uint8\_t [processIdentifier](#)
- uint8\_t [processIdMask](#)
- void \* [extension](#)

**Field Documentation****16.92.2.3.1 uint32\_t endAddr**

Memory region end address

Definition at line 154 of file mpu\_pal.h.

**16.92.2.3.2 void\* extension**

This field will be used to add extra settings to the basic region configuration

Definition at line 162 of file mpu\_pal.h.

**16.92.2.3.3 const mpu\_master\_access\_permission\_t\* masterAccRight**

Access permission for masters

Definition at line 155 of file mpu\_pal.h.

**16.92.2.3.4 uint8\_t processIdEnable**

Process identifier enable For MPU: the bit index corresponding with masters For SMPU: disable if equal zero, otherwise enable



Definition at line 156 of file mpu\_pal.h.

#### 16.92.2.3.5 uint8\_t processIdentifier

Process identifier

Definition at line 159 of file mpu\_pal.h.

#### 16.92.2.3.6 uint8\_t processIdMask

Process identifier mask. The setting bit will ignore the same bit in process identifier

Definition at line 160 of file mpu\_pal.h.

#### 16.92.2.3.7 uint32\_t startAddr

Memory region start address

Definition at line 153 of file mpu\_pal.h.

### 16.92.3 Typedef Documentation

#### 16.92.3.1 typedef mpu\_access\_rights\_t mpu\_access\_permission\_t

MPU detail access permission For specific master:

Code	Supervisor	User	Description
MPU_SUPERVISOR_↔ RWX_USER_NONE	r w x	- - -	Allow Read, write, execute in supervisor mode; no access in user mode
MPU_SUPERVISOR_↔ RWX_USER_X	r w x	- - x	Allow Read, write, execute in supervisor mode; execute in user mode
MPU_SUPERVISOR_↔ RWX_USER_W	r w x	- w -	Allow Read, write, execute in supervisor mode; write in user mode
MPU_SUPERVISOR_↔ RWX_USER_WX	r w x	- w x	Allow Read, write, execute in supervisor mode; write and execute in user mode
MPU_SUPERVISOR_↔ RWX_USER_R	r w x	r - -	Allow Read, write, execute in supervisor mode; read in user mode
MPU_SUPERVISOR_↔ RWX_USER_RX	r w x	r - x	Allow Read, write, execute in supervisor mode; read and execute in user mode
MPU_SUPERVISOR_↔ RWX_USER_RW	r w x	r w -	Allow Read, write, execute in supervisor mode; read and write in user mode

MPU_SUPERVISOR_↔ RWX_USER_RWX	r w x	r w x	Allow Read, write, execute in supervisor mode; read, write and execute in user mode
MPU_SUPERVISOR_↔ RX_USER_NONE	r - x	- - -	Allow Read, execute in supervisor mode; no access in user mode
MPU_SUPERVISOR_↔ RX_USER_X	r - x	- - x	Allow Read, execute in supervisor mode; execute in user mode
MPU_SUPERVISOR_↔ RX_USER_W	r - x	- w -	Allow Read, execute in supervisor mode; write in user mode
MPU_SUPERVISOR_↔ RX_USER_WX	r - x	- w x	Allow Read, execute in supervisor mode; write and execute in user mode
MPU_SUPERVISOR_↔ RX_USER_R	r - x	r - -	Allow Read, execute in supervisor mode; read in user mode
MPU_SUPERVISOR_↔ RX_USER_RX	r - x	r - x	Allow Read, execute in supervisor mode; read and execute in user mode
MPU_SUPERVISOR_↔ RX_USER_RW	r - x	r w -	Allow Read, execute in supervisor mode; read and write in user mode
MPU_SUPERVISOR_↔ RX_USER_RWX	r - x	r w x	Allow Read, execute in supervisor mode; read, write and execute in user mode
MPU_SUPERVISOR_↔ RW_USER_NONE	r w -	- - -	Allow Read, write in supervisor mode; no access in user mode
MPU_SUPERVISOR_↔ RW_USER_X	r w -	- - x	Allow Read, write in supervisor mode; execute in user mode
MPU_SUPERVISOR_↔ RW_USER_W	r w -	- w -	Allow Read, write in supervisor mode; write in user mode
MPU_SUPERVISOR_↔ RW_USER_WX	r w -	- w x	Allow Read, write in supervisor mode; write and execute in user mode
MPU_SUPERVISOR_↔ RW_USER_R	r w -	r - -	Allow Read, write in supervisor mode; read in user mode
MPU_SUPERVISOR_↔ RW_USER_RX	r w -	r - x	Allow Read, write in supervisor mode; read and execute in user mode

MPU_SUPERVISOR_↔ RW_USER_RW	r w -	r w -	Allow Read, write in supervisor mode; read and write in user mode
MPU_SUPERVISOR_↔ RW_USER_RWX	r w -	r w x	Allow Read, write in supervisor mode; read, write and execute in user mode
MPU_SUPERVISOR_↔ USER_NONE	- - -	- - -	No access allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_X	- - x	- - x	Execute operation is allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_W	- w -	- w -	Write operation is allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_WX	- w x	- w x	Write and execute operations are allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_R	r - -	r - -	Read operation is allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_RX	r - x	r - x	Read and execute operations are allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_RW	r w -	r w -	Read and write operations are allowed in user and supervisor modes
MPU_SUPERVISOR_↔ USER_RWX	r w x	r w x	Read write and execute operations are allowed in user and supervisor modes

For normal master:

Code	Read/Write permission	Description
MPU_NONE	- -	No Read/Write access permission
MPU_W	- w	Write access permission
MPU_R	r -	Read access permission
MPU_RW	r w	Read/Write access permission

Implements : mpu\_access\_permission\_t\_Class

Definition at line 126 of file mpu\_pal.h.

#### 16.92.4 Enumeration Type Documentation

##### 16.92.4.1 enum mpu\_error\_access\_type\_t

MPU access error Implements : mpu\_error\_access\_type\_t\_Class.

Enumerator

**MPU\_ERROR\_TYPE\_READ** Error type: read

**MPU\_ERROR\_TYPE\_WRITE** Error type: write

Definition at line 45 of file mpu\_pal.h.

## 16.92.4.2 enum mpu\_error\_attributes\_t

MPU access error attributes Implements : mpu\_error\_attributes\_t\_Class.

## Enumerator

**MPU\_ERROR\_USER\_MODE\_INSTRUCTION\_ACCESS** Instruction access error in user mode

**MPU\_ERROR\_USER\_MODE\_DATA\_ACCESS** Data access error in user mode

**MPU\_ERROR\_SUPERVISOR\_MODE\_INSTRUCTION\_ACCESS** Instruction access error in supervisor mode

**MPU\_ERROR\_SUPERVISOR\_MODE\_DATA\_ACCESS** Data access error in supervisor mode

Definition at line 55 of file mpu\_pal.h.

## 16.92.4.3 enum mpu\_inst\_type\_t

Enumeration with the types of peripherals supported by MPU PAL.

This enumeration contains the types of peripherals supported by MPU PAL. Implements : mpu\_inst\_type\_t\_Class

Definition at line 51 of file mpu\_pal\_mapping.h.

## 16.92.5 Function Documentation

## 16.92.5.1 status\_t MPU\_Deinit ( const mpu\_instance\_t \*const instance )

De-initializes memory protection unit by resetting all regions and masters to default and disable module.

## Parameters

in	instance	The pointer to MPU instance number.
----	----------	-------------------------------------

## Returns

operation status

- STATUS\_SUCCESS : Operation was successful.
- STATUS\_ERROR : Operation failed due to the region was locked by another master or all masters are locked.
- STATUS\_UNSUPPORTED : Operation was unsupported.

Definition at line 276 of file mpu\_pal.c.

## 16.92.5.2 status\_t MPU\_EnableRegion ( const mpu\_instance\_t \*const instance, uint8\_t regionNum, bool enable )

Enables or disables an exist region configuration.

## Parameters

in	instance	The pointer to MPU instance number.
in	regionNum	The region number.
in	enable	Valid state <ul style="list-style-type: none"> <li>• true : Enable region.</li> <li>• false : Disable region.</li> </ul>

**Returns**

operation status

- STATUS\_SUCCESS : Operation was successful.
- STATUS\_ERROR : Operation failed due to the region was locked by another master or all masters are locked.
- STATUS\_UNSUPPORTED : Operation was unsupported.

Definition at line 425 of file mpu\_pal.c.

**16.92.5.3** `status_t MPU_GetDefaultRegionConfig ( const mpu_instance_t *const instance, mpu_↔  
master_access_permission_t * masterAccRight, mpu_region_config_t * regionConfig  
)`

Gets default region configuration. Grants all access rights for masters; disable PID and cache; unlock region descriptor.

**Parameters**

in	<i>instance</i>	The pointer to MPU instance number.
out	<i>masterAccRight</i>	The pointer to master configuration structure, see <a href="#">mpu_master_access_↔ permission_t</a> . The length of array should be defined by number of masters supported by hardware.
out	<i>regionConfig</i>	The pointer to default region configuration structure, see <a href="#">mpu_region_config_↔ _t</a> .

**Returns**

operation status

- STATUS\_SUCCESS : Operation was successful.
- STATUS\_UNSUPPORTED : Operation was unsupported.

Definition at line 311 of file mpu\_pal.c.

**16.92.5.4** `bool MPU_GetError ( const mpu_instance_t *const instance, uint8_t channel, mpu_error_info_t * errPtr )`

Checks and gets the access error detail information then clear error flag if the error caused by a master.

**Parameters**

in	<i>instance</i>	The pointer to MPU instance number.
in	<i>channel</i>	The error capture channel For MPU: corresponding with the slave port number For SMPU: corresponding with the the master number
out	<i>errPtr</i>	The pointer to access error info structure, see <a href="#">mpu_error_info_t</a> .

**Returns**

operation status

- true : An error has occurred.
- false : No error has occurred or the operation was unsupported.

Definition at line 462 of file mpu\_pal.c.

**16.92.5.5** `status_t MPU_Init ( const mpu_instance_t *const instance, uint8_t regionCnt, const mpu_region_config_t *  
configPtr )`

Initializes memory protection unit by allocating regions and granting access rights for masters.

**Parameters**

in	<i>instance</i>	The pointer to MPU instance number.
in	<i>regionCnt</i>	The number of regions configured.
in	<i>configPtr</i>	The pointer to regions configuration structure, see <a href="#">mpu_region_config_t</a> .

**Returns**

operation status

- STATUS\_SUCCESS : Operation was successful.
- STATUS\_ERROR : Operation failed due to invalid master number or the region was locked by another master or all masters are locked.
- STATUS\_UNSUPPORTED : Operation was unsupported.

Definition at line 213 of file mpu\_pal.c.

**16.92.5.6** `status_t MPU_UpdateRegion ( const mpu_instance_t *const instance, uint8_t regionNum, const mpu_region_config_t * configPtr )`

Updates region configuration.

**Parameters**

in	<i>instance</i>	The pointer to MPU instance number.
in	<i>regionNum</i>	The region number.
in	<i>configPtr</i>	The pointer to region configuration structure, see <a href="#">mpu_region_config_t</a> .

**Returns**

operation status

- STATUS\_SUCCESS : Operation was successful.
- STATUS\_ERROR : Operation failed due to invalid master number or the region was locked by another master or all masters are locked.
- STATUS\_UNSUPPORTED : Operation was unsupported.

Definition at line 380 of file mpu\_pal.c.

## 16.93 Memory Protection Unit (MPU)

### 16.93.1 Detailed Description

The S32 SDK provides Peripheral Driver for the Memory Protection Unit (MPU) module of S32 SDK devices.

The memory protection unit (MPU) provides hardware access control for all memory references generated in the device.

#### Hardware background

The MPU concurrently monitors all system bus transactions and evaluates their appropriateness using pre-programmed region descriptors that define memory spaces and their access rights. Memory references that have sufficient access control rights are allowed to complete, while references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response.

The MPU implements a two-dimensional hardware array of memory region descriptors and the crossbar slave ports to continuously monitor the legality of every memory reference generated by each bus master in the system.

The feature set includes:

- 8(16 for S32K148) program-visible 128-bit region descriptors, accessible by four 32-bit words each
  - Each region descriptor defines a modulo-32 byte space, aligned anywhere in memory
    - \* Region sizes can vary from 32 bytes to 4 Gbytes
  - Two access control permissions defined in a single descriptor word
    - \* Masters 0–3: read, write, and execute attributes for supervisor and user accesses
    - \* Masters 4–7: read and write attributes
  - Hardware-assisted maintenance of the descriptor valid bit minimizes coherency issues
  - Alternate programming model view of the access control permissions word
  - Priority given to granting permission over denying access for overlapping region descriptors
- Detects access protection errors if a memory reference does not hit in any memory region, or if the reference is illegal in all hit memory regions. If an access error occurs, the reference is terminated with an error response, and the MPU inhibits the bus cycle being sent to the targeted slave device.
- Error registers, per slave port, capture the last faulting address, attributes, and other information
- Global MPU enable/disable control bit

#### Logical Bus Master Assignments and Possible Access Types

ID	Master	User	Super-visor	Data	Instruc-tion	Read	Write	Exe-cute	PID
0	Core	x	x	x	x	x	x	x	x
1	Debug-ger	x	x	x	x	x	x	x	x
2	DMA		x	x		x	x		
3	ENET	x		x		x	x		

ID	S32K1xx	S32MTV	S32K1xxW
0	x	x	x
1	x	x	x
2	x	x	x
3	x(1)		x(1)

1: S32K148 only.

#### Logical Slave Port Assignments

Port	Source	Destination
0	Crossbar slave port 0	Flash Controller
1	Crossbar slave port 1	SRAM backdoor
2	Code Bus	SRAM_L frontdoor
3	System Bus	SRAM_U frontdoor
4	Crossbar slave port 3	QuadSPI

Port	S32K11x	S32K14x	S32MTV	S32K1xxW
0	x	x	x	x
1	x(1)	x	x	x
2		x	x	x
3		x	x	x
4		x(2)		x(2)

1: Destination: SRAM controller/MTB/DWT/MCM. 2: S32K148 only.

## AHB-AP

AHB-AP provides the debugger access to all memory and registers in the system.

The MPU includes default settings and protections for the Region Descriptor 0 (RGD0) such that the Debugger always has access to the entire address space and those rights cannot be changed by the core or any other bus master.

## ERRATA

- S32K148:
  - E11109: The MPU requires a special programming sequence to protect the QSPI space as it is unable to see the two MSB bits of the QSPI address on slave port 4.  
This programming sequence requires 2 Region Descriptors [RGDx]:
    - \* One will cover the region 0x280x\_xxxx and the other one will cover region 0x680x\_xxxx.
    - \* When any master without permissions tries to access region 0x680x\_xxxx, an error will be captured in both, EDR3 and EDR4 registers. Moreover, the address of the failed access is captured on EAR3 and EAR4 registers. However, EAR3 will capture the address 0x680x\_xxxx, which is the one that belongs to the QSPI space. While EAR4 will capture the 0x280x\_xxxx address.

## Note

- S32K14x, S32K14xW:
  - In order to protect cache data, AHB LMEM will distribute all transactions to MPU slave port 2 for any access to the whole cacheable code bus memory domain.
    - \* If there is a Pflash access protection error by CM4, both slave port 0 & slave port 2 will report the same error.

## Modules

- [MPU Driver](#)  
*Memory Protection Unit Peripheral Driver.*



## 16.94 Memory Protection Unit Peripheral Abstraction Layer (MPU PAL)

### 16.94.1 Detailed Description

The S32 SDK provides a Peripheral Abstraction Layer for Memory Protection Unit (MPU) modules of S32 SDK devices.

The MPU PAL driver provides memory protection functionality via allocate regions and restrict access rights of all masters on the region. It was designed to be portable across all platforms and IPs which support Memory Protection Unit.

#### Integration guideline

Unlike the other drivers, MPU PAL modules need to include a configuration file named `mpu_pal_cfg.h`, which allows the user to specify which IPs are used and how many resources are allocated for each of them (state structures). The following code example shows how to configure one instance for each available MPU IPs.

```
#ifndef MPU_PAL_CFG_H
#define MPU_PAL_CFG_H

/* Define which IP instance which supported on this device */
#define MPU_OVER_MPU
#define MPU_OVER_SMPU

#endif /* MPU_PAL_CFG_H */
```

#### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\pal\src\mpu\mpu_pal.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\pal\inc\
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

[Memory Protection Unit \(MPU\)](#) smpu

#### IPs specification

The following tables contains IPs specification on platforms:

- Available

IP/MCU	S32K1xx	MPC574x	S32Rx7x
MPU	x		
SMPU		x	x

- Number of supported instances

IP/MCU	S32K1xx	MPC574x	S32Rx7x
MPU	1	–	–
SMPU	–	1(1)	2

1: 2 instances with MPC5747C, MPC5748C, MPC5746G, MPC5747G and MPC5748G.

SMPU instance	MPC574x	S32Rx7x
0	Flash and peripherals	Data XBAR (XBAR_0)
1	RAM	instruction XBAR (XBAR_1)

MPC574x: The cores on the device treats the following memory region as guarded by default:

- Guarded Region0: Address 0xFC00\_0000 to 0xFFFF\_FFFF
- Guarded Region1: Address 0xF800\_0000 to 0xFBFF\_FFFF

- Number of supported regions

IP/MCU	S32K1xx	MPC574x	S32Rx7x
MPU	8(1)	–	–
SMPU	–	16	16

1: 16 regions with S32K148.

#### Note

- S32K14x, S32K14xW:
  - In order to protect cache data, AHB LMEM will distribute all transactions to MPU slave port 2 for any access to the whole cacheable code bus memory domain.
  - \* If there is a Pflash access protection error by CM4, both slave port 0 & slave port 2 will report the same error.

#### Initialization & De-initialization

- In order to use the MPU PAL driver it must be first initialized, using [MPU\\_Init\(\)](#) function to initialize or re-initialize module.
- Example:

##### 1. Definitions for MPU IP (MPU\_OVER\_MPU)

```

/* Define MPU PAL instance */
mpu_instance_t mpu_pal_Instance =
{
    .instType = MPU_INST_TYPE_MPU, /* MPU PAL over MPU */
    .instIdx = 0U /* MPU instance 0 */
}

/* Define number of masters supported by platform */
#define MPU_PAL_MASTER_COUNT (16U)

/* Define number of used regions (should be in range supported by platform) */
#define MPU_PAL_REGION_COUNT (1U)

/* Status variable */
status_t status;

```

##### 2. Region configuration

```

/* Master configuration */
mpu_master_access_permission_t mpu_pal_masterAccRight[MPU_PAL_MASTER_COUNT] =
{
    /* Master */
    {
        .masterNum = FEATURE_MPU_MASTER_CORE, /* Core */
        .accessRight = MPU_ACCESS_SUPERVISOR_RWX_USER_RWX /* Access right: read, write and execute
                                                             for both supervisor and user mode */
    },
    /* Define the rest masters here */
    ...
};

/* Region configuration */
mpu_region_config_t mpu_pal_regionConfigs[MPU_PAL_REGION_COUNT] =
{
    /* Region 0 */
    {
        .startAddr = 0U, /* Start address */
        .endAddr = 0xFFFFFFFFU, /* End address */
    }
}

```

```

        .masterAccRight = mpu_pal_masterAccRight, /* Pointer to access right of all masters */
/* If support PID */
        .processIdEnable = 0x01U, /* 8'b00000001 Enable PID for logical master 0 (Core) */
        .processIdentifier = 0x00U, /* Process identifier */
        .processIdMask = 0xFFU, /* Process identifier mask */
/* End if */
/* Extension */
        .extension = NULL /* This field will be used to add extra settings
                           to the basic region configuration */
/* End extension */
    }
};

```

Or using `MPU_GetDefaultConfig()`

```

/* Master configuration */
mpu_master_access_permission_t mpu_pal_masterAccRight[MPU_PAL_MASTER_COUNT];
/* Region configuration */
mpu_region_config_t regionConfig0;

/* Get default region configuration */
status = MPU_GetDefaultConfig(&mpu_pal_Instance, mpu_pal_masterAccRight, &regionConfig0);
mpu_region_config_t mpu_pal_regionConfigs[MPU_PAL_REGION_COUNT] =
{
    regionConfig0
};

```

### 3. Initialization

```

/* Initializes MPU PAL */
status = MPU_Init(&mpu_pal_Instance, MPU_PAL_REGION_COUNT, mpu_pal_regionConfigs);

```

### 4. De-initialization

```

/* De-initializes MPU PAL */
status = MPU_Deinit(&mpu_pal_Instance);

```

## Updates region configuration

- The MPU PAL driver provides a function named `MPU_UpdateRegion()` to update a region configuration (address, access rights of all masters, process identifier,...).
- In order to remove unused region or add again, `MPU_EnableRegion()` can be used.
- Please note the region will be unlocked if the update succeed.
- Example:

#### 1. Modify (or add new) region after initialization

```

/* Disables process identifier functionality on region 0 */
regionConfig0.processIdEnable = 0x00U;

/* Updates region 0 */
status = MPU_UpdateRegion(&mpu_pal_Instance, 0U, &regionConfig0);

```

#### 2. Enables/Disables an exist region configuration

```

/* Enables region 1 */
status = MPU_EnableRegion(&mpu_pal_Instance, 1U, true);

/* Disables region 2 */
status = MPU_EnableRegion(&mpu_pal_Instance, 2U, false);

```

## Detects access protection errors

- The MPU PAL driver provides a function named `MPU_GetError()` to detect an access protection error on error capture channel. The channel can be different among IPs.
- Example:

```

/* Define error variable */
mpu_error_info_t mpu_pal_errVal;

/* Gets information on channel 0 */
bool errStatus = MPU_GetError(&mpu_pal_Instance, 0U, &mpu_pal_errVal)

```

## Other IP specific details

• **MPU** (MPU\_OVER\_MPU)

- Support PID for specific masters corresponding with the processIdEnable bit index.
- Detects an access error on slave ports:

Source	Slave port	Destination	S32K11x	S32K14x	S32MTV
Crossbar slave port 0	0	Flash Controller	x	x	x
Crossbar slave port 1	1	SRAM backdoor	x(1)	x	x
Code Bus	2	SRAM_L frontdoor		x	x
System Bus	3	SRAM_U frontdoor		x	x
Crossbar slave port 2	4	QuadSPI		x(2)	

1: Destination: SRAM controller/MTB/DWT/MCM.

2: S32K148 only.

• **SMPU** (MPU\_OVER\_SMPU)

- Support PID for for all specific masters (processIdEnable same as bool)
- Detects an access error on bus masters.
- Supports lock and cache inhibit features in region extension.
  - \* An address range specified in an MPU region descriptor for a cacheable space (that is, CI = 0) must be defined with a starting address aligned on a 0-modulo-32 byte address and with a multiple of the 32 byte cache line size factoring into the end address.
- MPU\_UpdateRegionLock() can be used to update lock configuration on a region.
- MPU\_GetRegionLockInfo() can be used to get lock status on a region.
- Example:

## 1. Extension

```

/* E.g. MPC5748G */
/* SMPU region extension with normal access rights */
mpu_over_smpu_extension mpu_pal_extension =
{
    /* If specific access supported */
    .specAccessEnable = false, /* Use normal access rights */
    .specAccessSet = NULL, /* Specific access configuration
                           Only use when specific access enabled */

    /* End if */
    .cacheInhibitEnable = true, /* The region cannot be cached */
    .lockConfig = MPU_UNLOCK /* The region is unlocked */
}

/* Use specific access rights */
#define MPU_PAL_REGION_ACCESS_SET_COUNT 3U /* Support 3 configurations on each region */
mpu_specific_access_permission_t mpu_pal_specificAccessConfig[MPU_PAL_REGION_ACCESS_SET_COUNT] =
{
    /* Set 1 */
    MPU_SUPERVISOR_RWX_USER_RWX, /* Allow read, write and execute for both
                                   supervisor and user mode */
    /* Set 2 */
    MPU_SUPERVISOR_RWX_USER_RWX, /* Allow read, write and execute for both
                                   supervisor and user mode */
    /* Set 3 */
    MPU_SUPERVISOR_RWX_USER_RWX /* Allow read, write and execute for both supervisor and user mode */
}

mpu_pal_extension.specAccessEnable = true;
mpu_pal_extension.specAccessSet = true;

/* SMPU Master configuration /
#define MPU_PAL_MASTER_COUNT 15U
mpu_master_access_permission_t mpu_pal_masterAccRight[MPU_PAL_MASTER_COUNT] =
{
    /* Master */
    {

```

```

        .masterNum    = FEATURE_SMPU_MASTER_CORE_Z4A, /* Core Z4A */
        .accessRight = MPU_RW_OR_SET_3              /* Normal access rights: read, write and execute for both
                                                    supervisor and user mode
                                                    Specific access: use set 3 in region configuration */
    },
    /* Define the rest masters here */
    ...
};

/* SMPU region configuration */
#define MPU_PAL_REGION_COUNT 1U
mpu_region_config_t mpu_pal_regionConfigs[MPU_PAL_REGION_COUNT] =
{
    /* Region 0 */
    {
        .startAddr = 0U,                      /* Start address */
        .endAddr = 0xFFFFFFFFFU,             /* End address */
        .masterAccRight = mpu_pal_masterAccRight, /* Pointer to access right of all masters */
        /* If support PID */
        .processIdEnable = true,              /* Enable process identifier for all masters */
        .processIdentifier = 0x00U,           /* Process identifier */
        .processIdMask = 0xFFU,              /* Process identifier mask */
        /* End if */
        /* Extension */
        .extension = &mpu_pal_extension      /* This field will be used to add extra settings
                                                    to the basic region configuration */
    }
}

/* Initialization */
...

```

## 2. Update lock configuration and get lock status on region

```

/* All masters cannot write to region descriptor 0 (cannot modify region 0 configuration) */
status = MPU_UpdateRegionLock(&mpu_pal_Instance, 0U, MPU_ALL_LOCK);

/* Gets lock status on region 0 */
mpu_region_lock_t lockStatus;
status = MPU_GetRegionLockInfo(&mpu_pal_Instance, 0U, &lockStatus);

```

## Modules

- [MPU PAL](#)

*Memory Protection Unit Peripheral Abstraction Layer.*

## 16.95 Node configuration

### 16.95.1 Detailed Description

This group contains APIs that used for node configuration purpose.

#### Functions

- `I_bool Id_is_ready_j2602 (I_ifc_handle iii)`  
*Verifies a state of node setting (using for J2602 and LIN 2.0).*
- `I_u8 Id_check_response_j2602 (I_ifc_handle iii, I_u8 *const RSID, I_u8 *const error_code)`  
*Verifies the state of response (using for J2602 and LIN 2.0) Master node only.*
- `I_bool Id_reconfig_msg_ID (I_ifc_handle iii, I_u8 dnn)`  
*This function reconfigures frame identifiers of a J2602 slave node based on input dnn.*
- `I_bool Id_assign_NAD_j2602 (I_ifc_handle iii, I_u8 dnn)`  
*This function assigns NAD of a J2602 slave device based on input DNN that is Device Node Number. NAD is (0x60+ DNN).*

### 16.95.2 Function Documentation

#### 16.95.2.1 I\_bool Id\_assign\_NAD\_j2602 ( I\_ifc\_handle iii, I\_u8 dnn )

This function assigns NAD of a J2602 slave device based on input DNN that is Device Node Number. NAD is (0x60+ DNN).

##### Parameters

<code>in</code>	<code>iii</code>	LIN interface handle
<code>in</code>	<code>dnn</code>	DNN of the device

##### Returns

- `I_bool`: 0: successful: New Configured NAD is 0x60 + DNN  
`I_bool`: 1: Unsuccessful: for either one of the following reasons:
- The protocol of this interface is not J2602
  - This device is a Master node in this interface
  - The input DNN is greater than 0xD that is invalid

Definition at line 1740 of file `lin_diagnostic_service.c`.

#### 16.95.2.2 I\_u8 Id\_check\_response\_j2602 ( I\_ifc\_handle iii, I\_u8 \*const RSID, I\_u8 \*const error\_code )

Verifies the state of response (using for J2602 and LIN 2.0) Master node only.

##### Parameters

<code>in</code>	<code>iii</code>	LIN interface handle
<code>out</code>	<code>RSID</code>	buffer for saving the response ID
<code>out</code>	<code>error_code</code>	buffer for saving the error code

##### Returns

`I_u8` status of the last service

Definition at line 1528 of file `lin_diagnostic_service.c`.

#### 16.95.2.3 I\_bool Id\_is\_ready\_j2602 ( I\_ifc\_handle iii )

Verifies a state of node setting (using for J2602 and LIN 2.0).

**Parameters**

<i>in</i>	<i>iii</i>	LIN interface handle
-----------	------------	----------------------

**Returns**

*I\_bool*

Definition at line 1502 of file `lin_diagnostic_service.c`.

**16.95.2.4 *I\_bool* `Id_reconfig_msg_ID` ( *I\_ifc\_handle iii*, *I\_u8 dnn* )**

This function reconfigures frame identifiers of a J2602 slave node based on input *dnn*.

**Parameters**

<i>in</i>	<i>iii</i>	LIN interface handle
<i>in</i>	<i>dnn</i>	DNN of the device

**Returns**

*I\_bool*: 0: successful: Frame Identifiers were reconfigured based on input DNN according to NAD Message ID mapping table.

*I\_bool*: 1: Unsuccessful: for either one of the following reasons:

- The protocol of this interface is not J2602
- This device is a Master node in this interface
- The input DNN is greater than 0xD that is invalid
- The slave has more than 16 configurable frames
- The slave has 9-16 configurable frames, and *dnn* is 0xC or 0xD
- The slave has 5-8 configurable frames, and *dnn* is not 0x00, 0x2, 0x4, 0x6, 0x8, 0xA, 0xC.

Definition at line 1622 of file `lin_diagnostic_service.c`.

## 16.96 Node configuration

### 16.96.1 Detailed Description

This group contains APIs that used for node configuration purpose.

With protocol lin2.1 in slave node, some service like (Data dump, Conditional change nad with id from 2 to 255) are not supported by LinStack but user can implement it in application by use function `Id_receive_message` and `Id_send_message` in transport layer.

With protocol J2602 in slave node, some service like (Data dump, Assign NAD, Conditional change NAD) are not supported by LinStack but user can implement it in application by choosing these services in supported\_sid in PEX GUI and use function `Id_receive_message` and `Id_send_message` in transport layer. When received target reset master request slave node just update `status_byte` and send response positive message.

#### Functions

- `I_u8 Id_is_ready (I_ifc_handle iii)`  
*This call returns the status of the last requested configuration service.*
- `void Id_check_response (I_ifc_handle iii, I_u8 *const RSID, I_u8 *const error_code)`  
*This call returns the result of the last node configuration service, in the parameters RSID and error\_code. A value in RSID is always returned but not always in the error\_code. Default values for RSID and error\_code is 0 (zero).*
- `void Id_assign_frame_id_range (I_ifc_handle iii, I_u8 NAD, I_u8 start_index, const I_u8 *const PIDs)`  
*This function assigns the protected identifier of up to four frames.*
- `void Id_save_configuration (I_ifc_handle iii, I_u8 NAD)`  
*This function to issue a save configuration request to a slave node.*
- `I_u8 Id_read_configuration (I_ifc_handle iii, I_u8 *const data, I_u8 *const length)`  
*This function copies current configuration in a reserved area.*
- `I_u8 Id_set_configuration (I_ifc_handle iii, const I_u8 *const data, I_u16 length)`  
*This function configures slave node according to data.*
- `void Id_assign_NAD (I_ifc_handle iii, I_u8 initial_NAD, I_u16 supplier_id, I_u16 function_id, I_u8 new_NAD)`  
*This call assigns the NAD (node diagnostic address) of all slave nodes that matches the initial\_NAD, the supplier ID and the function ID. Master node only.*
- `void Id_conditional_change_NAD (I_ifc_handle iii, I_u8 NAD, I_u8 id, I_u8 byte_data, I_u8 mask, I_u8 invert, I_u8 new_NAD)`  
*This call changes the NAD if the node properties fulfill the test specified by id, byte, mask and invert. Master node only.*

### 16.96.2 Function Documentation

#### 16.96.2.1 void Id\_assign\_frame\_id\_range ( I\_ifc\_handle iii, I\_u8 NAD, I\_u8 start\_index, const I\_u8 \*const PIDs )

This function assigns the protected identifier of up to four frames.

##### Parameters

in	<i>iii</i>	lin interface handle
in	<i>NAD</i>	Node address value of the target node
in	<i>start_index</i>	specifies which is the first frame to assign a PID
in	<i>PIDs</i>	list of protected identifier

##### Returns

void

This API is available for master interfaces only

Definition at line 149 of file `lin_diagnostic_service.c`.



16.96.2.2 void Id\_assign\_NAD( l\_ifc\_handle *iii*, l\_u8 *initial\_NAD*, l\_u16 *supplier\_id*, l\_u16 *function\_id*, l\_u8 *new\_NAD* )

This call assigns the NAD (node diagnostic address) of all slave nodes that matches the initial\_NAD, the supplier ID and the function ID. Master node only.

**Parameters**

in	<i>iii</i>	LIN interface handle
in	<i>initial_NAD</i>	Initial node address of the target node
in	<i>supplier_id</i>	Supplier ID of the target node
in	<i>function_id</i>	Function identifier of the target node
in	<i>new_NAD</i>	New node address

**Returns**

void

This call assigns the NAD (node diagnostic address) of all slave nodes that matches the initial\_NAD, the supplier ID and the function ID. The new NAD of the slave node will be new\_NAD. This function is used for master node only.

Definition at line 845 of file lin\_diagnostic\_service.c.

### 16.96.2.3 void Id\_check\_response ( I\_ifc\_handle *iii*, I\_u8 \*const *RSID*, I\_u8 \*const *error\_code* )

This call returns the result of the last node configuration service, in the parameters RSID and error\_code. A value in RSID is always returned but not always in the error\_code. Default values for RSID and error\_code is 0 (zero).

For slave interfaces Id\_check\_response shall do nothing

**Parameters**

in	<i>iii</i>	lin interface handle
out	<i>RSID</i>	buffer for saving the response ID
out	<i>error_code</i>	buffer for saving the error code

This API is available for master interfaces only

Definition at line 118 of file lin\_diagnostic\_service.c.

### 16.96.2.4 void Id\_conditional\_change\_NAD ( I\_ifc\_handle *iii*, I\_u8 *NAD*, I\_u8 *id*, I\_u8 *byte\_data*, I\_u8 *mask*, I\_u8 *invert*, I\_u8 *new\_NAD* )

This call changes the NAD if the node properties fulfill the test specified by id, byte, mask and invert. Master node only.

**Parameters**

in	<i>iii</i>	:LIN interface handle
in	<i>NAD</i>	Current NAD value of the target node
in	<i>id</i>	Property ID of the target node
in	<i>byte</i>	Byte location of property value to be read from the target node
in	<i>mask</i>	Value for masking the read property byte
in	<i>invert</i>	Value for excluding the read property byte
in	<i>new_NAD</i>	New NAD value to be assigned when the condition is met

**Returns**

void

This call changes the NAD if the node properties fulfill the test specified by id, byte, mask and invert.

Definition at line 886 of file lin\_diagnostic\_service.c.

### 16.96.2.5 I\_u8 Id\_is\_ready ( I\_ifc\_handle *iii* )

This call returns the status of the last requested configuration service.

**Parameters**

<i>in</i>	<i>iii</i>	lin interface handle
-----------	------------	----------------------

**Returns**

LD\_SERVICE\_BUSY Service is ongoing.

LD\_REQUEST\_FINISHED The configuration request has been completed. This is a intermediate status between the configuration request and configuration response.

LD\_SERVICE\_IDLE The configuration request/response combination has been completed, i.e. the response is valid and may be analyzed. Also, this value is returned if no request has yet been called.

LD\_SERVICE\_ERROR The configuration request or response experienced an error. Error here means error on the bus, and not a negative configuration response from the slave node.

Definition at line 92 of file lin\_diagnostic\_service.c.

#### 16.96.2.6 I\_u8 ld\_read\_configuration ( I\_ifc\_handle *iii*, I\_u8 \*const *data*, I\_u8 \*const *length* )

This function copies current configuration in a reserved area.

**Parameters**

<i>in</i>	<i>iii</i>	Lin interface handle
<i>out</i>	<i>data</i>	Data area to save configuration,
<i>out</i>	<i>length</i>	Length of data area (1 + n, NAD + PIDs)

**Returns**

LD\_READ\_OK If the service was successful.

LD\_LENGTH\_TOO\_SHORT If the configuration size is greater than the length. It means that the data area does not contain a valid configuration.

This function is implemented Slave Only. Set the expected length value to EXP = NN + NF, where : NN = the number of NAD. NF = the number of configurable frames; Moreover: Not taken PID's diagnostics frame: 3C, 3D

Definition at line 446 of file lin\_diagnostic\_service.c.

#### 16.96.2.7 void ld\_save\_configuration ( I\_ifc\_handle *iii*, I\_u8 *NAD* )

This function to issue a save configuration request to a slave node.

**Parameters**

<i>in</i>	<i>iii</i>	Interface name
<i>in</i>	<i>NAD</i>	Node address of target

**Returns**

void

This function is available for master nodes only. This function is available for all diagnostic classes and only for LIN2.1 and above. This function is called to send a save configuration request to a specific slave node with the given NAD, or to all slave nodes if NAD is set to broadcast This function is implemented for Master Only.

Definition at line 191 of file lin\_diagnostic\_service.c.

#### 16.96.2.8 I\_u8 ld\_set\_configuration ( I\_ifc\_handle *iii*, const I\_u8 \*const *data*, I\_u16 *length* )

This function configures slave node according to data.

**Parameters**

<i>in</i>	<i>iii</i>	Lin interface handle
<i>in</i>	<i>data</i>	Structure containing the NAD and all the n PIDs for the frames of the specified NAD,
<i>in</i>	<i>length</i>	Length of data area (1 + n, NAD + PIDs)

**Returns**

LD\_SET\_OK If the service was successful

LD\_LENGTH\_NOT\_CORRECT If the required size of the configuration is not equal to the given length.

LD\_DATA\_ERROR The set of configuration could not be made.

This function is implemented Slave Only. Set the expected length value to  $EXP = NN + NF$ , where : NN = the number of NAD. NF = the number of configurable frames; Moreover: Not taken PID's diagnostics frame: 3C, 3D

Definition at line 511 of file lin\_diagnostic\_service.c.

## 16.97 Node identification

### 16.97.1 Detailed Description

This group contains API that used for node identification purpose.

Read by identifier service just support id 0 and 1. User can implement for other id by modify function `Id_read_by_id`↔`_id_callout` in generated file `lin_cfg.c`.

#### Functions

- void `Id_read_by_id` ( `I_ifc_handle` *iii*, `I_u8` *NAD*, `I_u16` *supplier\_id*, `I_u16` *function\_id*, `I_u8` *id*, `lin_product_id_t` *\*const data* )

*The call requests the slave node selected with the NAD to return the property associated with the id parameter. Master node only.*

### 16.97.2 Function Documentation

**16.97.2.1** void `Id_read_by_id` ( `I_ifc_handle` *iii*, `I_u8` *NAD*, `I_u16` *supplier\_id*, `I_u16` *function\_id*, `I_u8` *id*, `lin_product_id_t` *\*const data* )

The call requests the slave node selected with the NAD to return the property associated with the id parameter. Master node only.

#### Parameters

in	<i>iii</i>	LIN interface handle
in	<i>NAD</i>	Value of the target node
in	<i>supplier_id</i>	Supplier ID of the target node
in	<i>function_id</i>	Function ID of the target node
in	<i>id</i>	ID of the target node
out	<i>data</i>	Buffer for saving the data read from the node

#### Returns

void

The call requests the slave node selected with the NAD to return the property associated with the id parameter.

Definition at line 933 of file `lin_diagnostic_service.c`.

## 16.98 Notification

This group contains APIs that let users know when a signal's value changed.

## 16.99 OS Interface (OSIF)

### 16.99.1 Detailed Description

#### OS Interface Layer (OSIF)

The OSIF layer is a minimal wrapper layer for common RTOS services, intended to be used by SDK drivers and middlewares. It can be used by the user application, but it is not recommended. The operations supported by OSIF:

- mutex lock/unlock
- semaphore post/wait
- time delay and get time elapsed

OSIF currently comes in two variants: bare-metal and FreeRTOS. Steps to use each one are described below.

#### FreeRTOS

To integrate the FreeRTOS OSIF variant, two steps are necessary:

- compile and link the `osif_freertos.c` file
- define a project-wide compile symbol:

```
USING_OS_FREERTOS
```

#### FreeRTOSConfig.h dependencies

FreeRTOS configuration file needs to have these options activated (set to 1 in FreeRTOSConfig.h):

- `INCLUDE_xQueueGetMutexHolder`
- `INCLUDE_xTaskGetCurrentTaskHandle`

#### Hardware resources

FreeRTOS OSIF uses the FreeRTOS API and services, does not use any additional hardware or software resources.

#### FreeRTOS supported platforms

The SDK platforms supported by FreeRTOS can be found in the following table. If a platform is supported by FreeRTOS, both osif variants, bare-metal and freertos, are supported. If the platform is not supported by FreeRTOS, only osif bare-metal variant is applicable:

Platform	FreeRTOS support
S32K11x	Yes
S32K14x	Yes
S32K14xW	Yes
MPC5746C	Yes
MPC5748G	Yes
MPC5744P	Yes
S32R274	Yes
S32R372	Yes

#### FreeRTOS static vs dynamic memory allocation

OSIF objects will use static memory allocation schemes (FreeRTOS 9.0.0 and above) if the feature is enabled.

This should be transparent for the upper layer which uses OSIF.

The FreeRTOS configuration option is

```
configSUPPORT_STATIC_ALLOCATION
```

If it's set to 1, the OSIF will use static memory allocation.

## Bare-metal

To integrate the bare-metal OSIF variant:

- compile and link the [osif\\_baremetal.c](#) file

Mutex operations are dummy operations (always return success) and semaphore is implemented as a simple counter.

## Hardware resources

Bare-metal OSIF uses a hardware timer to accurately measure time. The timer and channel used are platform-dependent, are chosen to be the same as the FreeRTOS implementation if possible.

The table below shows which timers and channels are used on each platform:

Platform	Timer	Channel
S32K11x	Systick	N/A
S32K14x	Systick	N/A
S32K14xW	Systick	N/A
MPC5746C	PIT	15
MPC5748G	PIT	15
MPC5744P	PIT	3
S32R274	PIT0	3
S32R372	PIT0	3

## Bare-metal timing limitations

For bare-metal OSIF, the timer is initialized at the first call in OSIF that needs timing. That is either [OSIF\\_TimeDelay](#), [OSIF\\_MutexLock](#) or [OSIF\\_SemaWait](#) (functions with timeout). The timer configuration, but not the counter, is updated at each subsequent call to these functions.

Do not assume [OSIF\\_GetMilliseconds](#) will return a global value since system initialization. It will return the global value since the very first timer initialization, mentioned above.

## Integration guideline

### Compilation units

The following files need to be compiled in the project:

- for baremetal variant:

```
${S32SDK_PATH}\rtos\osif\osif_baremetal.c
```

- for FreeRTOS variant:

```
${S32SDK_PATH}\rtos\osif\osif_freertos.c
```

### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\rtos\osif\
```

### Compile symbols

- for baremetal variant: No special symbols are required for this component
- for FreeRTOS variant: The following symbol must be added to the compile symbols of the toolchain:

```
USING_OS_FREERTOS
```



## Dependencies

- for baremetal variant:

[Clock Manager](#)[Interrupt Manager \(Interrupt\)](#)

- for FreeRTOS variant:

[FreeRTOS](#)

## Macros

- #define [OSIF\\_WAIT\\_FOREVER](#) 0xFFFFFFFFu

## Functions

- void [OSIF\\_TimeDelay](#) (const uint32\_t delay)  
*Delays execution for a number of milliseconds.*
- uint32\_t [OSIF\\_GetMilliseconds](#) (void)  
*Returns the number of milliseconds elapsed since starting the internal timer or starting the scheduler.*
- status\_t [OSIF\\_MutexLock](#) (const mutex\_t \*const pMutex, const uint32\_t timeout)  
*Waits for a mutex and locks it.*
- status\_t [OSIF\\_MutexUnlock](#) (const mutex\_t \*const pMutex)  
*Unlocks a previously locked mutex.*
- status\_t [OSIF\\_MutexCreate](#) (mutex\_t \*const pMutex)  
*Create an unlocked mutex.*
- status\_t [OSIF\\_MutexDestroy](#) (const mutex\_t \*const pMutex)  
*Destroys a previously created mutex.*
- status\_t [OSIF\\_SemaWait](#) (semaphore\_t \*const pSem, const uint32\_t timeout)  
*Decrement a semaphore with timeout.*
- status\_t [OSIF\\_SemaPost](#) (semaphore\_t \*const pSem)  
*Increment a semaphore.*
- status\_t [OSIF\\_SemaCreate](#) (semaphore\_t \*const pSem, const uint8\_t initValue)  
*Creates a semaphore with a given value.*
- status\_t [OSIF\\_SemaDestroy](#) (const semaphore\_t \*const pSem)  
*Destroys a previously created semaphore.*

## 16.99.2 Macro Definition Documentation

16.99.2.1 #define [OSIF\\_WAIT\\_FOREVER](#) 0xFFFFFFFFu

Definition at line 71 of file osif.h.

## 16.99.3 Function Documentation

16.99.3.1 uint32\_t [OSIF\\_GetMilliseconds](#) ( void )

Returns the number of milliseconds elapsed since starting the internal timer or starting the scheduler.

## Returns

the number of milliseconds elapsed

Definition at line 233 of file osif\_baremetal.c.

### 16.99.3.2 `status_t OSIF_MutexCreate ( mutex_t *const pMutex )`

Create an unlocked mutex.

**Parameters**

<i>in</i>	<i>pMutex</i>	reference to the mutex object
-----------	---------------	-------------------------------

**Returns**

One of the possible status codes:

- STATUS\_SUCCESS: mutex created
- STATUS\_ERROR: mutex could not be created

Definition at line 281 of file osif\_baremetal.c.

**16.99.3.3 status\_t OSIF\_MutexDestroy ( const mutex\_t \*const pMutex )**

Destroys a previously created mutex.

**Parameters**

<i>in</i>	<i>pMutex</i>	reference to the mutex object
-----------	---------------	-------------------------------

**Returns**

One of the possible status codes:

- STATUS\_SUCCESS: mutex destroyed

Definition at line 295 of file osif\_baremetal.c.

**16.99.3.4 status\_t OSIF\_MutexLock ( const mutex\_t \*const pMutex, const uint32\_t timeout )**

Waits for a mutex and locks it.

**Parameters**

<i>in</i>	<i>pMutex</i>	reference to the mutex object
<i>in</i>	<i>timeout</i>	time-out value in milliseconds

**Returns**

One of the possible status codes:

- STATUS\_SUCCESS: mutex lock operation success
- STATUS\_ERROR: mutex already owned by current thread
- STATUS\_TIMEOUT: mutex lock operation timed out

Definition at line 251 of file osif\_baremetal.c.

**16.99.3.5 status\_t OSIF\_MutexUnlock ( const mutex\_t \*const pMutex )**

Unlocks a previously locked mutex.

**Parameters**

<i>in</i>	<i>pMutex</i>	reference to the mutex object
-----------	---------------	-------------------------------

**Returns**

One of the possible status codes:

- STATUS\_SUCCESS: mutex unlock operation success
- STATUS\_ERROR: mutex unlock failed

Definition at line 267 of file osif\_baremetal.c.

16.99.3.6 `status_t OSIF_SemaCreate ( semaphore_t *const pSem, const uint8_t initValue )`

Creates a semaphore with a given value.

**Parameters**

in	<i>pSem</i>	reference to the semaphore object
in	<i>initValue</i>	initial value of the semaphore

**Returns**

One of the possible status codes:

- STATUS\_SUCCESS: semaphore created
- STATUS\_ERROR: semaphore could not be created

Definition at line 402 of file osif\_baremetal.c.

**16.99.3.7 status\_t OSIF\_SemaDestroy ( const semaphore\_t \*const pSem )**

Destroys a previously created semaphore.

**Parameters**

in	<i>pSem</i>	reference to the semaphore object
----	-------------	-----------------------------------

**Returns**

One of the possible status codes:

- STATUS\_SUCCESS: semaphore destroyed

Definition at line 420 of file osif\_baremetal.c.

**16.99.3.8 status\_t OSIF\_SemaPost ( semaphore\_t \*const pSem )**

Increment a semaphore.

**Parameters**

in	<i>pSem</i>	reference to the semaphore object
----	-------------	-----------------------------------

**Returns**

One of the possible status codes:

- STATUS\_SUCCESS: semaphore post operation success
- STATUS\_ERROR: semaphore could not be incremented

Definition at line 375 of file osif\_baremetal.c.

**16.99.3.9 status\_t OSIF\_SemaWait ( semaphore\_t \*const pSem, const uint32\_t timeout )**

Decrement a semaphore with timeout.

**Parameters**

in	<i>pSem</i>	reference to the semaphore object
in	<i>timeout</i>	time-out value in milliseconds

**Returns**

One of the possible status codes:

- STATUS\_SUCCESS: semaphore wait operation success
- STATUS\_TIMEOUT: semaphore wait timed out

Definition at line 311 of file osif\_baremetal.c.

16.99.3.10 void OSIF\_TimeDelay ( const uint32\_t *delay* )

Delays execution for a number of milliseconds.

## Parameters

<code>in</code>	<code>delay</code>	Time delay in milliseconds.
-----------------	--------------------	-----------------------------

Definition at line 208 of file `osif_baremetal.c`.

## 16.100 Output Compare - Peripheral Abstraction Layer (OC PAL)

### 16.100.1 Detailed Description

The S32 SDK provides a Peripheral Abstraction Layer for the output compare mode of S32 SDK devices.

The OC PAL driver allows to set pin, clear pin or toggle pin. It was designed to be portable across all platforms and IPs which support FTM, eMIOS, FlexPWM and eTIMER.

#### How to integrate OC PAL in your application

Unlike the other drivers, OC PAL modules need to include a configuration file named `oc_pal_cfg.h`. This one allows the user to specify which IPs are used. The following code example shows how to configure one instance for each available OC IPs.

```
#ifndef OC_PAL_CFG_H
#define OC_PAL_CFG_H

/* Define which IP instance will be used in current project */
#define OC_PAL_OVER_EMIOS

#endif /* OC_PAL_CFG_H */
```

The following table contains the matching between platforms and available IPs

IP/ M↔ CU	S32↔ K116	S32↔ K118	S32↔ K142	S32↔ K142↔ W	S32↔ K144	S32↔ K144↔ W	S32↔ K146	S32↔ K148	M↔ P↔ C5748 G	M↔ P↔ C5746 C	M↔ P↔ C5744 P	S32↔ R274	S32↔ R372
FT↔ M↔ _↔ OC	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	NO	NO	NO	NO	NO
e↔ MI↔ O↔ S_↔ OC	NO	NO	NO	NO	NO	NO	NO	NO	Y↔ ES	Y↔ ES	NO	NO	NO
eT↔ IM↔ ER	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	Y↔ ES	Y↔ ES	Y↔ ES
Flex↔ P↔ WM	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	Y↔ ES	Y↔ ES	Y↔ ES

#### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\pal\src\oc\oc_pal.c
${S32SDK_PATH}\platform\pal\src\oc\oc_irq.c
${S32SDK_PATH}\platform\pal\src\oc\oc_irq.h
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\pal\inc\
```

#### Compile symbols

No special symbols are required for this component



## Dependencies

[Clock Manager](#) [Interrupt Manager \(Interrupt\)](#) [ftm\\_oc](#) [emios\\_oc](#) [etimer](#) [flexpwm](#)

## Features

- Set the output signal can be set, cleared, or toggled pin
- Start/stop the channel in the output compare mode
- Force the channel output to high or low level

## Functionality

### Initialization

In order to use the OC PAL driver it must be first initialized, using [OC\\_Init\(\)](#) function. Once initialized, it should be de-initialized before initialized again for the same OC module instance, using [OC\\_Deinit\(\)](#). The initialization function does the following operations:

- sets the clock source, clock prescaler
- sets the number of channel output compare are used
- configures the channel output to set or clear or toggle pin

### Example:

```

const oc_instance_t oc_pall_instance = { OC_INST_TYPE_ETIMER, 0U };

channel_extension_etimer_for_oc_t oc_pall_etimerChnExtension0 =
{
    .primaryInput =
    {
        .source = ETIMER_IN_SRC_CLK_DIV_128,
        .polarity = ETIMER_POLARITY_POSITIVE,
    },
    .outputPin =
    {
        .enable = true,
        .polarity = ETIMER_POLARITY_POSITIVE,
    },
};

oc_output_ch_param_t oc_pall_ChnConfig[1] =
{
    /* Channel configuration 0 */
    {
        .hwChannelId      = 4U,
        .chMode            = OC_TOGGLE_ON_MATCH,
        .comparedValue     = 62500,
        .channelExtension  = &oc_pall_etimerChnExtension0,
        .channelCallbackParams = NULL,
        .channelCallbacks  = oc_pall_channel_callBack0
    }
};

oc_config_t oc_pall_InitConfig =
{
    .numChannels      = 1U,
    .outputChConfig   = oc_pall_ChnConfig,
    .extension        = NULL
};

/* Initialize output compare mode */
OC_Init(&oc_pall_instance, &oc_pall_InitConfig);

```

### De-initialize a output compare instance

This function will disable the output compare mode. The driver can't be used again until reinitialized. All register are reset to default value and counter is stopped.

**Example:**

```
/* De-initialize output compare mode */
OC_Deinit(&oc_pall_instance);
```

**Start the channel in the output compare mode**

This function will set the channel is in the output compare mode.

**Example:**

```
uint8_t hwChannel = oc_pall_InitConfig.outputChConfig->hwChannelId;

/* Start channel in the output compare mode */
OC_StartChannel(&oc_pall_instance, hwChannel);
```

**Stop the channel in the output compare mode**

This function will set the channel is used in GPIO mode or other peripheral.

**Example:**

```
uint8_t hwChannel = oc_pall_InitConfig.outputChConfig->hwChannelId;

/* Stop channel in the output compare mode */
OC_StopChannel(&oc_pall_instance, hwChannel);
```

**Control the channel output by software**

This function is used to forces the output pin to a specified value. It can be used to control the output pin value when the OC channel is disabled.

**Example:**

```
uint8_t hwChannel = oc_pall_InitConfig.outputChConfig->hwChannelId;

/* Force the channel output by software */
OC_SetOutputState(&oc_pall_instance, hwChannel, false);
```

**Set the operation mode of channel output**

This function will set the action executed on a compare match value to set output pin, clear output pin, toggle output pin.

**Example:**

```
uint8_t hwChannel = oc_pall_InitConfig.outputChConfig->hwChannelId;

/* Change the channel output to toggle pin */
OC_SetOutputAction(&oc_pall_instance, hwChannel,
    OC_TOGGLE_ON_MATCH);
```

**Update the match value on the channel**

This function will update the value of an output compare channel to the counter matches to this value.

**Example:**

```
uint8_t hwChannel = oc_pall_InitConfig.outputChConfig->hwChannelId;

/* Set the match counter to new value */
OC_SetCompareValue(&oc_pall_instance, hwChannel, 0x1000UL,
    OC_RELATIVE_VALUE);
```

**Important Notes**

- Before using the OC PAL driver the module clock must be configured. Refer to Clock Manager for clock configuration.

- The board specific configurations must be done prior to driver after that can call APIs.
- Some features are not available for all OC IPs and incorrect parameters will be handled by DEV\_ASSERT.

### Data Structures

- struct `oc_output_ch_param_t`  
The configuration structure of output compare parameters for each channel. [More...](#)
- struct `oc_config_t`  
Defines the configuration structures are used in the output compare mode. [More...](#)
- struct `extension_ftm_for_oc_t`  
Defines the extension structure for the output compare mode over FTM. [More...](#)

### Enumerations

- enum `oc_option_mode_t` { `OC_DISABLE_OUTPUT` = 0x00U, `OC_TOGGLE_ON_MATCH` = 0x01U, `OC_CLEAR_ON_MATCH` = 0x02U, `OC_SET_ON_MATCH` = 0x03U }  
The type of comparison for output compare mode Implements : `oc_option_mode_t` Class.
- enum `oc_option_update_t` { `OC_RELATIVE_VALUE` = 0x00U, `OC_ABSOLUTE_VALUE` = 0x01U }  
The type of update on the channel match Implements : `oc_option_update_t` Class.

### Functions

- status\_t `OC_Init` (const `oc_instance_t` \*const instance, const `oc_config_t` \*const configPtr)  
Initializes the output compare mode.
- status\_t `OC_Deinit` (const `oc_instance_t` \*const instance)  
De-initialize the output compare instance.
- void `OC_StartChannel` (const `oc_instance_t` \*const instance, const uint8\_t channel)  
Start the counter.
- void `OC_StopChannel` (const `oc_instance_t` \*const instance, const uint8\_t channel)  
Stop the counter.
- status\_t `OC_SetOutputState` (const `oc_instance_t` \*const instance, const uint8\_t channel, bool outputValue)  
Control the channel output by software.
- status\_t `OC_SetOutputAction` (const `oc_instance_t` \*const instance, const uint8\_t channel, `oc_option_mode_t` channelMode)  
Set the operation mode of channel output.
- status\_t `OC_SetCompareValue` (const `oc_instance_t` \*const instance, const uint8\_t channel, uint32\_t nextCompareMatchValue, `oc_option_update_t` typeOfupdate)  
Update the match value on the channel.
- void `OC_EnableNotification` (const `oc_instance_t` \*const instance, const uint8\_t channel)  
Enable channel notifications.
- void `OC_DisableNotification` (const `oc_instance_t` \*const instance, const uint8\_t channel)  
Disable channel notifications.

## 16.100.2 Data Structure Documentation

### 16.100.2.1 struct oc\_output\_ch\_param\_t

The configuration structure of output compare parameters for each channel.

Implements : `oc_output_ch_param_t` Class

Definition at line 83 of file `oc_pal.h`.

**Data Fields**

- [uint8\\_t hwChannelId](#)
- [oc\\_option\\_mode\\_t chMode](#)
- [uint16\\_t comparedValue](#)
- [void \\* channelExtension](#)
- [void \\* channelCallbackParams](#)
- [oc\\_callback\\_t channelCallbacks](#)

**Field Documentation****16.100.2.1.1 void\* channelCallbackParams**

The parameter of callback application for channels event

Definition at line 89 of file oc\_pal.h.

**16.100.2.1.2 oc\_callback\_t channelCallbacks**

The callback function for channels event

Definition at line 90 of file oc\_pal.h.

**16.100.2.1.3 void\* channelExtension**

The IP specific configuration structure for channel

Definition at line 88 of file oc\_pal.h.

**16.100.2.1.4 oc\_option\_mode\_t chMode**

Channel output mode

Definition at line 86 of file oc\_pal.h.

**16.100.2.1.5 uint16\_t comparedValue**

The compared value

Definition at line 87 of file oc\_pal.h.

**16.100.2.1.6 uint8\_t hwChannelId**

Physical hardware channel ID

Definition at line 85 of file oc\_pal.h.

**16.100.2.2 struct oc\_config\_t**

Defines the configuration structures are used in the output compare mode.

Implements : [oc\\_config\\_t\\_Class](#)

Definition at line 98 of file oc\_pal.h.

**Data Fields**

- [uint8\\_t nNumChannels](#)
- [const oc\\_output\\_ch\\_param\\_t \\* outputChConfig](#)
- [void \\* extension](#)

**Field Documentation****16.100.2.2.1 void\* extension**

IP specific configuration structure

Definition at line 102 of file oc\_pal.h.

#### 16.100.2.2.2 uint8\_t nNumChannels

Number of output compare channel used

Definition at line 100 of file oc\_pal.h.

#### 16.100.2.2.3 const oc\_output\_ch\_param\_t\* outputChConfig

Output compare channels configuration

Definition at line 101 of file oc\_pal.h.

#### 16.100.2.3 struct extension\_ftm\_for\_oc\_t

Defines the extension structure for the output compare mode over FTM.

Part of FTM configuration structure Implements : extension\_ftm\_for\_oc\_t\_Class

Definition at line 112 of file oc\_pal.h.

#### Data Fields

- uint16\_t maxCountValue
- ftm\_clock\_source\_t ftmClockSource
- ftm\_clock\_ps\_t ftmPrescaler

#### Field Documentation

##### 16.100.2.3.1 ftm\_clock\_source\_t ftmClockSource

Select clock source for FTM

Definition at line 115 of file oc\_pal.h.

##### 16.100.2.3.2 ftm\_clock\_ps\_t ftmPrescaler

Register pre-scaler options available in the ftm\_clock\_ps\_t enumeration

Definition at line 116 of file oc\_pal.h.

##### 16.100.2.3.3 uint16\_t maxCountValue

Maximum count value in ticks

Definition at line 114 of file oc\_pal.h.

#### 16.100.3 Enumeration Type Documentation

##### 16.100.3.1 enum oc\_option\_mode\_t

The type of comparison for output compare mode Implements : oc\_option\_mode\_t\_Class.

#### Enumerator

- OC\_DISABLE\_OUTPUT** No action on output pin
- OC\_TOGGLE\_ON\_MATCH** Toggle on match
- OC\_CLEAR\_ON\_MATCH** Clear on match
- OC\_SET\_ON\_MATCH** Set on match

Definition at line 60 of file oc\_pal.h.

### 16.100.3.2 enum oc\_option\_update\_t

The type of update on the channel match Implements : oc\_option\_update\_t\_Class.

#### Enumerator

**OC\_RELATIVE\_VALUE** Next compared value is relative to current value

**OC\_ABSOLUTE\_VALUE** Next compared value is absolute

Definition at line 72 of file oc\_pal.h.

### 16.100.4 Function Documentation

#### 16.100.4.1 status\_t OC\_Deinit ( const oc\_instance\_t \*const instance )

De-initialize the output compare instance.

This function will disable the output compare mode. The driver can't be used again until reinitialized. The context structure is no longer needed by the driver and can be freed after calling this function.

#### Parameters

in	instance	The output compare instance number.
----	----------	-------------------------------------

#### Returns

Operation status

- STATUS\_SUCCESS: Operation was successful

Definition at line 1126 of file oc\_pal.c.

#### 16.100.4.2 void OC\_DisableNotification ( const oc\_instance\_t \*const instance, const uint8\_t channel )

Disable channel notifications.

This function disables channel notification.

#### Parameters

in	instance	The output compare instance number
in	channel	The channel number

Definition at line 1706 of file oc\_pal.c.

#### 16.100.4.3 void OC\_EnableNotification ( const oc\_instance\_t \*const instance, const uint8\_t channel )

Enable channel notifications.

This function enables channel notification.

#### Parameters

in	instance	The output compare instance number
in	channel	The channel number

Definition at line 1672 of file oc\_pal.c.

#### 16.100.4.4 status\_t OC\_Init ( const oc\_instance\_t \*const instance, const oc\_config\_t \*const configPtr )

Initializes the output compare mode.

This function will initialize the OC PAL instance, including the other platform specific HW units used together in the output compare mode. This function configures a group of channels in instance to set, clear toggle the output signal.

## Parameters

in	<i>instance</i>	The output compare instance number.
in	<i>configPtr</i>	The pointer to configuration structure.

## Returns

Operation status

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 1062 of file oc\_pal.c.

**16.100.4.5** `status_t OC_SetCompareValue ( const oc_instance_t *const instance, const uint8_t channel, uint32_t nextCompareMatchValue, oc_option_update_t typeOfupdate )`

Update the match value on the channel.

This function will update the value of an output compare channel to the counter matches to this value.

## Parameters

in	<i>instance</i>	The output compare instance number.
in	<i>channel</i>	The channel number.
in	<i>nextCompareMatchValue</i>	The timer value in ticks until the next compare match event should be appeared.
in	<i>typeOfupdate</i>	The type of update: <ul style="list-style-type: none"> <li>• OC_RELATIVE_VALUE : nextCompareMatchValue will be added to current counter value into the channel value register</li> <li>• OC_ABSOLUTE_VALUE : nextCompareMatchValue will be written into the channel value register</li> </ul>

## Returns

Operation status

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 1556 of file oc\_pal.c.

**16.100.4.6** `status_t OC_SetOutputAction ( const oc_instance_t *const instance, const uint8_t channel, oc_option_mode_t channelMode )`

Set the operation mode of channel output.

This function will set the action executed on a compare match value to set output pin, clear output pin, toggle output pin.

**Parameters**

in	<i>instance</i>	The output compare instance number.
in	<i>channel</i>	The channel number.
in	<i>channelMode</i>	The channel mode in output compare: <ul style="list-style-type: none"> <li>• OC_DISABLE_OUTPUT : No action on output pin</li> <li>• OC_TOGGLE_ON_MATCH : Toggle on match</li> <li>• OC_CLEAR_ON_MATCH : Clear on match</li> <li>• OC_SET_ON_MATCH : Set on match</li> </ul>

**Returns**

Operation status

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 1463 of file oc\_pal.c.

**16.100.4.7** `status_t OC_SetOutputState ( const oc_instance_t *const instance, const uint8_t channel, bool outputValue )`

Control the channel output by software.

This function is used to forces the output pin to a specified value.

**Parameters**

in	<i>instance</i>	The output compare instance number.
in	<i>channel</i>	The channel number.
in	<i>outputValue</i>	The output value: <ul style="list-style-type: none"> <li>• false : The software output control forces 0 to the channel output.</li> <li>• true : The software output control forces 1 to the channel output.</li> </ul>

**Returns**

Operation status

- STATUS\_SUCCESS : Completed successfully.
- STATUS\_ERROR : Error occurred.

Definition at line 1389 of file oc\_pal.c.

**16.100.4.8** `void OC_StartChannel ( const oc_instance_t *const instance, const uint8_t channel )`

Start the counter.

This function start channel counting.

**Parameters**

in	<i>instance</i>	The output compare instance number.
in	<i>channel</i>	The channel number.

Definition at line 1240 of file oc\_pal.c.

**16.100.4.9** `void OC_StopChannel ( const oc_instance_t *const instance, const uint8_t channel )`

Stop the counter.



This function stop channel output.

**Parameters**

<i>in</i>	<i>instance</i>	The output compare instance number.
<i>in</i>	<i>channel</i>	The channel number.

Definition at line 1323 of file oc\_pal.c.

## 16.101 PDB Driver

### 16.101.1 Detailed Description

Programmable Delay Block Peripheral Driver.

#### 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 pulse out to the CMP and the PDB counter.

The PDB contains a counter whose output is compared to several different digital values. If the PDB is enabled, then a trigger input event will reset the counter and make it start to count. A trigger input event is defined as a rising edge being detected on a selected trigger input source, or if software trigger is selected and a software trigger is issued. Each PDB channel is associated with 1 ADC block, and each PDB channel contains 8 pre-triggers. A pre-trigger has a delay associated and is mapped to an ADC channel; when the PDB counter is equal to the delay value configured for a pre-trigger, the pre-trigger is activated and selects the ADC channel that starts the conversion. Inside a PDB channel, the pre-triggers can be configured to work as chains, meaning that a pre-trigger is enabled automatically when the ADC conversion selected by the previous pre-trigger, completes its execution. The pre-trigger delays and back-to-back operation must be configured, in such a way that at most one pre-trigger per PDB channel is activated at any given point - otherwise a PDB sequence error occurs. This feature is called back-to-back mode, and needs to be configured individually for each pre-trigger using [PDB\\_DRV\\_ConfigAdcPreTrigger\(\)](#). On some devices (depending on availability), chains can be configured between different PDB channels or PDB instances - this can be configured using `interchannelBackToBackEnable` or `instanceBackToBackEnable`.

#### PDB Initialization

For initializing the PDB counter, input triggers or general pre-trigger settings (instance or interchannel back-to-back modes) call [PDB\\_DRV\\_Init\(\)](#). Note that all pre-triggers share the same counter. \* The basic timing/counting step is set when initializing the main PDB counter: The basic timing/counting step =  $F_{BusClkHz} / \text{pdb\_timer\_config\_t} \leftarrow \text{clkPreDiv} / \text{pdb\_timer\_config\_t} \leftarrow \text{clkPreMultFactor}$

The  $F_{BusClkHz}$  is the frequency of bus clock in Hertz. The "clkPreDiv" and "clkPreMultFactor" are in the [pdb\\_↔ timer\\_config\\_t](#) structure. All pre-triggering delays use this frequency.

#### Integration guideline

##### Compilation units

The following files need to be compiled in the project:

```
* ${S32SDK_PATH}\platform\drivers\src\pdb\pdb_driver.c
* ${S32SDK_PATH}\platform\drivers\src\pdb\pdb_hw_access.c
*
```

##### Include path

The following paths need to be added to the include path of the toolchain:

```
* ${S32SDK_PATH}\platform\drivers\inc\
* ${S32SDK_PATH}\platform\drivers\src\pdb\
*
```

##### Compile symbols

No special symbols are required for this component

##### Dependencies

[Clock Manager Interrupt Manager \(Interrupt\)](#)

## PDB Call diagram

Three kinds of typical use cases are designed for the PDB module.

- Normal Timer/Counter - the basic use-case. The Timer/Counter starts after the PDB is initialized, when the input trigger (hardware or software) is asserted. When the counter reaches the compare value (set via modulus register), an interrupt or DMA transfer occur if enabled. In continuous mode, when the counter reaches the modulus compare value, it resets to zero and starts counting again.
- Triggering for ADC module. Depending on the number of ADC channels that need to be used, each pre-trigger must be configured using `PDB_DRV_ConfigAdcPreTrigger()`. The PDB counter starts counting when the input trigger (hardware or software) is asserted. If the pre-trigger is enabled in normal mode (back-to-back disabled), it will be activated (thus starting the corresponding ADC channel conversion) when the PDB counter is equal with the configured pre-trigger delay. If the pre-trigger is enabled in back-to-back mode, it will trigger the corresponding ADC channel conversion, when the ADC channel corresponding to the previous pre-trigger in the chain, completes the conversion (ADC COCO flag set).
- Generate pulse outputs of configurable width. The Pulse-Out is set to high/low when the PDB counter reaches the values set in POyDLY[DLY1/2], configured via `PDB_DRV_SetCmpPulseOutDelayForHigh()` or `PDB_DRV_SetCmpPulseOutDelayForLow`. The Pulse-Out signal is connected to TRGMUX, which can route it to any other peripheral (e.g. can be used as a sample window for any CMP module).

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

### Normal Timer/Counter:

```
#define PDB_INSTANCE    OUL

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      =
        PDB_LOAD_VAL_IMMEDIATELY;
    PdbTimerConfig.seqErrIntEnable     = false;
    PdbTimerConfig.clkPreDiv           = PDB_CLK_PREDIV_BY_8;
    PdbTimerConfig.clkPreMultFactor    =
        PDB_CLK_PREMULT_FACT_AS_40;
    PdbTimerConfig.triggerInput        = PDB_SOFTWARE_TRIGGER;
    PdbTimerConfig.continuousModeEnable = true;
    PdbTimerConfig.dmaEnable           = false;
    PdbTimerConfig.intEnable           = true;
    PdbTimerConfig.instanceBackToBackEnable = false;
    PdbTimerConfig.interchannelBackToBackEnable = false;
    PDB_DRV_Init(instance, &PdbTimerConfig);
    PDB_DRV_SetTimerModulusValue(instance, 0xFFFFU);
    PDB_DRV_SetValueForTimerInterrupt(instance, 0xFFU);
    PDB_DRV_LoadValuesCmd(instance);
    gPdbIntCounter = 0U;
    gPdbInstance = instance;
    PDB_DRV_SoftTriggerCmd(instance);
    while (gPdbIntCounter < 20U) {}
    PRINTF("PDB Timer's delay interrupt generated.\r\n");
    PDB_DRV_Deinit(instance);
    PRINTF("OK.\r\n");
}

void PDB_IRQHandler()
{
    PDB_DRV_ClearTimerIntFlag(PDB_INSTANCE);
    if (gPdbIntCounter >= 0xFFFFU)
    {
        gPdbIntCounter = 0U;
    }
    else
    {
        gPdbIntCounter++;
    }
}

#if PDB_INSTANCE < 1
void PDB0_IRQHandler(void)
{
    PDB_IRQHandler();
}
```

```

    }
    #elif PDB_INSTANCE < 2
        void PDB1_IRQHandler(void)
        {
            PDB_IRQHandler();
        }
    #endif

```

### Trigger for ADC module:

```

void PDB_TEST_AdcPreTrigger(uint32_t instance)
{
    pdb_timer_config_t PdbTimerConfig;
    pdb_adc_pretrigger_config_t PdbAdcPreTriggerConfig;
    PdbTimerConfig.loadValueMode =
        PDB_LOAD_VAL_IMMEDIATELY;
    PdbTimerConfig.seqErrIntEnable = false;
    PdbTimerConfig.clkPreDiv = PDB_CLK_PREDIV_BY_8;
    PdbTimerConfig.clkPreMultFactor =
        PDB_CLK_PREMULT_FACT_AS_40;
    PdbTimerConfig.triggerInput = PDB_SOFTWARE_TRIGGER;
    PdbTimerConfig.continuousModeEnable = false;
    PdbTimerConfig.dmaEnable = false;
    PdbTimerConfig.intEnable = false;
    PdbTimerConfig.instanceBackToBackEnable = false;
    PdbTimerConfig.interchannelBackToBackEnable = 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\\_timer\\_config\\_t](#)  
Defines the type of structure for basic timer in PDB. [More...](#)
- struct [pdb\\_adc\\_pretrigger\\_config\\_t](#)  
Defines the type of structure for configuring ADC's pre\_trigger. [More...](#)

### Enumerations

- enum [pdb\\_load\\_value\\_mode\\_t](#) { PDB\_LOAD\_VAL\_IMMEDIATELY = 0U, PDB\_LOAD\_VAL\_AT\_MODULE\_COUNTER = 1U, PDB\_LOAD\_VAL\_AT\_NEXT\_TRIGGER = 2U, PDB\_LOAD\_VAL\_AT\_MODULE\_COUNTER\_OR\_NEXT\_TRIGGER = 3U }  
Defines the type of value load mode for the PDB module.
- enum [pdb\\_clk\\_prescaler\\_div\\_t](#) { PDB\_CLK\_PREDIV\_BY\_1 = 0U, PDB\_CLK\_PREDIV\_BY\_2 = 1U, PDB\_CLK\_PREDIV\_BY\_4 = 2U, PDB\_CLK\_PREDIV\_BY\_8 = 3U, PDB\_CLK\_PREDIV\_BY\_16 = 4U, PDB\_CLK\_PREDIV\_BY\_32 = 5U, PDB\_CLK\_PREDIV\_BY\_64 = 6U, PDB\_CLK\_PREDIV\_BY\_128 = 7U }  
Defines the type of prescaler divider for the PDB counter clock. Implements : [pdb\\_clk\\_prescaler\\_div\\_t\\_Class](#).
- enum [pdb\\_trigger\\_src\\_t](#) { PDB\_TRIGGER\_IN0 = 0U, PDB\_SOFTWARE\_TRIGGER = 15U }  
Defines the type of trigger source mode for the PDB.
- enum [pdb\\_clk\\_prescaler\\_mult\\_factor\\_t](#) { PDB\_CLK\_PREMULT\_FACT\_AS\_1 = 0U, PDB\_CLK\_PREMULT\_FACT\_AS\_10 = 1U, PDB\_CLK\_PREMULT\_FACT\_AS\_20 = 2U, PDB\_CLK\_PREMULT\_FACT\_AS\_40 = 3U }  
Defines the type of the multiplication source mode for PDB.

## Functions

- void [PDB\\_DRV\\_Init](#) (const uint32\_t instance, const [pdb\\_timer\\_config\\_t](#) \*userConfigPtr)  
*Initializes the PDB counter and triggers input.*
- void [PDB\\_DRV\\_Deinit](#) (const uint32\_t instance)  
*De-initializes the PDB module.*
- void [PDB\\_DRV\\_GetDefaultConfig](#) ([pdb\\_timer\\_config\\_t](#) \*const config)  
*Gets the default configuration structure of PDB with default settings.*
- void [PDB\\_DRV\\_Enable](#) (const uint32\_t instance)  
*Enables the PDB module.*
- void [PDB\\_DRV\\_Disable](#) (const uint32\_t instance)  
*Disables the PDB module.*
- void [PDB\\_DRV\\_SoftTriggerCmd](#) (const uint32\_t instance)  
*Triggers the PDB with a software trigger.*
- uint32\_t [PDB\\_DRV\\_GetTimerValue](#) (const uint32\_t instance)  
*Gets the current value of the PDB counter.*
- bool [PDB\\_DRV\\_GetTimerIntFlag](#) (const uint32\_t instance)  
*Gets the PDB interrupt flag.*
- void [PDB\\_DRV\\_ClearTimerIntFlag](#) (const uint32\_t instance)  
*Clears the interrupt flag.*
- void [PDB\\_DRV\\_LoadValuesCmd](#) (const uint32\_t instance)  
*Executes the command of loading values.*
- void [PDB\\_DRV\\_SetTimerModulusValue](#) (const uint32\_t instance, const uint16\_t value)  
*Sets the value of timer modulus.*
- void [PDB\\_DRV\\_SetValueForTimerInterrupt](#) (const uint32\_t instance, const uint16\_t value)  
*Sets the value for the timer interrupt.*
- void [PDB\\_DRV\\_ConfigAdcPreTrigger](#) (const uint32\_t instance, const uint32\_t chn, const [pdb\\_adc\\_pretrigger\\_config\\_t](#) \*configPtr)  
*Configures the ADC pre\_trigger in the PDB module.*
- uint32\_t [PDB\\_DRV\\_GetAdcPreTriggerFlags](#) (const uint32\_t instance, const uint32\_t chn, const uint32\_t preChnMask)  
*Gets the ADC pre\_trigger flag in the PDB module.*
- void [PDB\\_DRV\\_ClearAdcPreTriggerFlags](#) (const uint32\_t instance, const uint32\_t chn, const uint32\_t preChnMask)  
*Clears the ADC pre\_trigger flag in the PDB module.*
- uint32\_t [PDB\\_DRV\\_GetAdcPreTriggerSeqErrFlags](#) (const uint32\_t instance, const uint32\_t chn, const uint32\_t preChnMask)  
*Gets the ADC pre\_trigger flag in the PDB module.*
- void [PDB\\_DRV\\_ClearAdcPreTriggerSeqErrFlags](#) (const uint32\_t instance, const uint32\_t chn, const uint32\_t preChnMask)  
*Clears the ADC pre\_trigger flag in the PDB module.*
- void [PDB\\_DRV\\_SetAdcPreTriggerDelayValue](#) (const uint32\_t instance, const uint32\_t chn, const uint32\_t preChn, const uint32\_t value)  
*Sets the ADC pre\_trigger delay value in the PDB module.*
- void [PDB\\_DRV\\_SetCmpPulseOutEnable](#) (const uint32\_t instance, const uint32\_t pulseChnMask, bool enable)  
*Switches on/off the CMP pulse out in the PDB module.*
- void [PDB\\_DRV\\_SetCmpPulseOutDelayForHigh](#) (const uint32\_t instance, const uint32\_t pulseChn, const uint32\_t value)  
*Sets the CMP pulse out delay value for high in the PDB module.*
- void [PDB\\_DRV\\_SetCmpPulseOutDelayForLow](#) (const uint32\_t instance, const uint32\_t pulseChn, const uint32\_t value)  
*Sets the CMP pulse out delay value for low in the PDB module.*

### 16.101.2 Data Structure Documentation

#### 16.101.2.1 struct pdb\_timer\_config\_t

Defines the type of structure for basic timer in PDB.

Definition at line 103 of file pdb\_driver.h.

##### Data Fields

- [pdb\\_load\\_value\\_mode\\_t](#) loadValueMode
- bool [seqErrIntEnable](#)
- [pdb\\_clk\\_prescaler\\_div\\_t](#) clkPreDiv
- [pdb\\_clk\\_prescaler\\_mult\\_factor\\_t](#) clkPreMultFactor
- [pdb\\_trigger\\_src\\_t](#) triggerInput
- bool [continuousModeEnable](#)
- bool [dmaEnable](#)
- bool [intEnable](#)

##### Field Documentation

#### 16.101.2.1.1 [pdb\\_clk\\_prescaler\\_div\\_t](#) clkPreDiv

Select the prescaler divider.

Definition at line 107 of file pdb\_driver.h.

#### 16.101.2.1.2 [pdb\\_clk\\_prescaler\\_mult\\_factor\\_t](#) clkPreMultFactor

Select multiplication factor for prescaler.

Definition at line 108 of file pdb\_driver.h.

#### 16.101.2.1.3 bool [continuousModeEnable](#)

Enable the continuous mode.

Definition at line 110 of file pdb\_driver.h.

#### 16.101.2.1.4 bool [dmaEnable](#)

Enable the dma for timer.

Definition at line 111 of file pdb\_driver.h.

#### 16.101.2.1.5 bool [intEnable](#)

Enable the interrupt for timer. : interrupt is generated only if DMA is disabled.

Definition at line 112 of file pdb\_driver.h.

#### 16.101.2.1.6 [pdb\\_load\\_value\\_mode\\_t](#) loadValueMode

Select the load mode.

Definition at line 105 of file pdb\_driver.h.

#### 16.101.2.1.7 bool [seqErrIntEnable](#)

Enable PDB Sequence Error Interrupt.

Definition at line 106 of file pdb\_driver.h.

#### 16.101.2.1.8 `pdb_trigger_src_t` triggerInput

Select the trigger input source.

Definition at line 109 of file `pdb_driver.h`.

#### 16.101.2.2 `struct pdb_adc_pretrigger_config_t`

Defines the type of structure for configuring ADC's `pre_trigger`.

Definition at line 132 of file `pdb_driver.h`.

##### Data Fields

- `uint32_t` `adcPreTriggerIdx`
- `bool` `preTriggerEnable`
- `bool` `preTriggerOutputEnable`
- `bool` `preTriggerBackToBackEnable`

##### Field Documentation

#### 16.101.2.2.1 `uint32_t` `adcPreTriggerIdx`

Setting `pre_trigger`'s index.

Definition at line 134 of file `pdb_driver.h`.

#### 16.101.2.2.2 `bool` `preTriggerBackToBackEnable`

Enable the back to back mode for ADC `pre_trigger`. If enabled, the pretrigger will be activated automatically when the ADC COCO flag corresponding to the previous pretrigger in the chain, is set. The previous pretrigger for pretriggers with index 0, depend on features instanceBackToBackEnable and interchannelBackToBackEnable.

Definition at line 137 of file `pdb_driver.h`.

#### 16.101.2.2.3 `bool` `preTriggerEnable`

Enable the `pre_trigger`.

Definition at line 135 of file `pdb_driver.h`.

#### 16.101.2.2.4 `bool` `preTriggerOutputEnable`

Enable the `pre_trigger` output.

Definition at line 136 of file `pdb_driver.h`.

### 16.101.3 Enumeration Type Documentation

#### 16.101.3.1 `enum pdb_clk_prescaler_div_t`

Defines the type of prescaler divider for the PDB counter clock. Implements : `pdb_clk_prescaler_div_t_Class`.

##### Enumerator

- `PDB_CLK_PREDIV_BY_1`** Counting divided by 1 x prescaler multiplication factor (selected by MULT).
- `PDB_CLK_PREDIV_BY_2`** Counting divided by 2 x prescaler multiplication factor (selected by MULT).
- `PDB_CLK_PREDIV_BY_4`** Counting divided by 4 x prescaler multiplication factor (selected by MULT).
- `PDB_CLK_PREDIV_BY_8`** Counting divided by 8 x prescaler multiplication factor (selected by MULT).
- `PDB_CLK_PREDIV_BY_16`** Counting divided by 16 x prescaler multiplication factor (selected by MULT).
- `PDB_CLK_PREDIV_BY_32`** Counting divided by 32 x prescaler multiplication factor (selected by MULT).



**PDB\_CLK\_PREDIV\_BY\_64** Counting divided by 64 x prescaler multiplication factor (selected by MULT).

**PDB\_CLK\_PREDIV\_BY\_128** Counting divided by 128 x prescaler multiplication factor (selected by MULT).

Definition at line 58 of file pdb\_driver.h.

#### 16.101.3.2 enum pdb\_clk\_prescaler\_mult\_factor\_t

Defines the type of the multiplication source mode for PDB.

Selects the multiplication factor of the prescaler divider for the PDB counter clock. Implements : `pdb_clk_prescaler_mult_factor_t_Class`

##### Enumerator

**PDB\_CLK\_PREMULT\_FACT\_AS\_1** Multiplication factor is 1.

**PDB\_CLK\_PREMULT\_FACT\_AS\_10** Multiplication factor is 10.

**PDB\_CLK\_PREMULT\_FACT\_AS\_20** Multiplication factor is 20.

**PDB\_CLK\_PREMULT\_FACT\_AS\_40** Multiplication factor is 40.

Definition at line 89 of file pdb\_driver.h.

#### 16.101.3.3 enum pdb\_load\_value\_mode\_t

Defines the type of value load mode for the PDB module.

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. Implements : `pdb_load_value_mode_t_Class`

##### Enumerator

**PDB\_LOAD\_VAL\_IMMEDIATELY** Loaded immediately after load operation.

**PDB\_LOAD\_VAL\_AT\_MODULO\_COUNTER** Loaded when counter hits the modulo after load operation.

**PDB\_LOAD\_VAL\_AT\_NEXT\_TRIGGER** Loaded when detecting an input trigger after load operation.

**PDB\_LOAD\_VAL\_AT\_MODULO\_COUNTER\_OR\_NEXT\_TRIGGER** Loaded when counter hits the modulo or detecting an input trigger after load operation.

Definition at line 42 of file pdb\_driver.h.

#### 16.101.3.4 enum pdb\_trigger\_src\_t

Defines the type of trigger source mode for the PDB.

Selects the trigger input source for the PDB. The trigger input source can be internal or the software trigger. Implements : `pdb_trigger_src_t_Class`

##### Enumerator

**PDB\_TRIGGER\_IN0** Source trigger comes from TRGMUX.

**PDB\_SOFTWARE\_TRIGGER** Select software trigger.

Definition at line 77 of file pdb\_driver.h.

#### 16.101.4 Function Documentation

**16.101.4.1 void PDB\_DRV\_ClearAdcPreTriggerFlags ( const uint32\_t instance, const uint32\_t chn, const uint32\_t preChnMask )**

Clears the ADC pre\_trigger flag in the PDB module.

This function clears the ADC pre\_trigger flags in the PDB module.

**Parameters**

in	<i>instance</i>	PDB instance ID.
in	<i>chn</i>	PDB channel.
in	<i>preChnMask</i>	ADC pre_trigger channels mask.

Definition at line 379 of file pdb\_driver.c.

**16.101.4.2** void PDB\_DRV\_ClearAdcPreTriggerSeqErrFlags ( const uint32\_t *instance*, const uint32\_t *chn*, const uint32\_t *preChnMask* )

Clears the ADC pre\_trigger flag in the PDB module.

This function clears the ADC pre\_trigger sequence error flags in the PDB module.

**Parameters**

in	<i>instance</i>	PDB instance ID.
in	<i>chn</i>	PDB channel.
in	<i>preChnMask</i>	ADC pre_trigger channels mask.

Definition at line 415 of file pdb\_driver.c.

**16.101.4.3** void PDB\_DRV\_ClearTimerIntFlag ( const uint32\_t *instance* )

Clears the interrupt flag.

This function clears the interrupt flag.

**Parameters**

in	<i>instance</i>	PDB instance ID.
----	-----------------	------------------

Definition at line 278 of file pdb\_driver.c.

**16.101.4.4** void PDB\_DRV\_ConfigAdcPreTrigger ( const uint32\_t *instance*, const uint32\_t *chn*, const pdb\_adc\_pretrigger\_config\_t \* *configPtr* )

Configures the ADC pre\_trigger in the PDB module.

This function configures the ADC pre\_trigger in the PDB module. Important note: any pretrigger which is enabled and has the trigger output enabled (preTriggerOutputEnable and preTriggerEnable both true) must have the corresponding delay value set to a non-zero value by calling [PDB\\_DRV\\_SetAdcPreTriggerDelayValue](#) function.

**Parameters**

in	<i>instance</i>	PDB instance ID.
in	<i>chn</i>	PDB channel.
in	<i>configPtr</i>	Pointer to the user configuration structure. See <a href="#">pdb_adc_pretrigger_config_t</a> .

Definition at line 340 of file pdb\_driver.c.

**16.101.4.5** void PDB\_DRV\_Deinit ( const uint32\_t *instance* )

De-initializes the PDB module.

This function de-initializes the PDB module. Calling this function shuts down the PDB module and reduces the power consumption.

**Parameters**

in	<i>instance</i>	PDB instance ID.
----	-----------------	------------------

Definition at line 136 of file pdb\_driver.c.

**16.101.4.6 void PDB\_DRV\_Disable ( const uint32\_t *instance* )**

Disables the PDB module.

This function disables the PDB module, counter is off also.

**Parameters**

<i>in</i>	<i>instance</i>	PDB instance ID.
-----------	-----------------	------------------

Definition at line 215 of file pdb\_driver.c.

**16.101.4.7 void PDB\_DRV\_Enable ( const uint32\_t *instance* )**

Enables the PDB module.

This function enables the PDB module, counter is on.

**Parameters**

<i>in</i>	<i>instance</i>	PDB instance ID.
-----------	-----------------	------------------

Definition at line 200 of file pdb\_driver.c.

**16.101.4.8 uint32\_t PDB\_DRV\_GetAdcPreTriggerFlags ( const uint32\_t *instance*, const uint32\_t *chn*, const uint32\_t *preChnMask* )**

Gets the ADC pre\_trigger flag in the PDB module.

This function gets the ADC pre\_trigger flags in the PDB module.

**Parameters**

<i>in</i>	<i>instance</i>	PDB instance ID.
<i>in</i>	<i>chn</i>	PDB channel.
<i>in</i>	<i>preChnMask</i>	ADC pre_trigger channels mask.

**Returns**

Assertion of indicated flag.

Definition at line 361 of file pdb\_driver.c.

**16.101.4.9 uint32\_t PDB\_DRV\_GetAdcPreTriggerSeqErrFlags ( const uint32\_t *instance*, const uint32\_t *chn*, const uint32\_t *preChnMask* )**

Gets the ADC pre\_trigger flag in the PDB module.

This function gets the ADC pre\_trigger flags in the PDB module.

**Parameters**

<i>in</i>	<i>instance</i>	PDB instance ID.
<i>in</i>	<i>chn</i>	PDB channel.
<i>in</i>	<i>preChnMask</i>	ADC pre_trigger channels mask.

**Returns**

Assertion of indicated flag.

Definition at line 397 of file pdb\_driver.c.

**16.101.4.10 void PDB\_DRV\_GetDefaultConfig ( pdb\_timer\_config\_t \*const *config* )**

Gets the default configuration structure of PDB with default settings.

This function initializes the hardware configuration structure to default values (Reference Manual Resets). This function should be called before configuring the hardware feature by `PDB_DRV_Init()` function, otherwise all members be written by user. This function ensures that all members are written with safe values, but the user still can modify the desired members.

#### Parameters

<code>out</code>	<code>config</code>	Pointer to PDB configuration structure.
------------------	---------------------	---

Definition at line 164 of file `pdb_driver.c`.

#### 16.101.4.11 `bool PDB_DRV_GetTimerIntFlag ( const uint32_t instance )`

Gets the PDB interrupt flag.

This function gets the PDB interrupt flag. It is asserted if the PDB interrupt occurs.

#### Parameters

<code>in</code>	<code>instance</code>	PDB instance ID.
-----------------	-----------------------	------------------

#### Returns

Assertion of indicated event.

Definition at line 263 of file `pdb_driver.c`.

#### 16.101.4.12 `uint32_t PDB_DRV_GetTimerValue ( const uint32_t instance )`

Gets the current value of the PDB counter.

This function gets the current counter value.

#### Parameters

<code>in</code>	<code>instance</code>	PDB instance ID.
-----------------	-----------------------	------------------

#### Returns

Current PDB counter value.

Definition at line 247 of file `pdb_driver.c`.

#### 16.101.4.13 `void PDB_DRV_Init ( const uint32_t instance, const pdb_timer_config_t * userConfigPtr )`

Initializes the PDB counter and triggers input.

This function initializes the PDB counter, input triggers and general pre-trigger settings. 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

<code>in</code>	<code>instance</code>	PDB instance ID.
<code>in</code>	<code>userConfigPtr</code>	Pointer to the user configuration structure. See the " <code>pdb_user_config_t</code> ".

Definition at line 61 of file `pdb_driver.c`.

#### 16.101.4.14 `void PDB_DRV_LoadValuesCmd ( const uint32_t instance )`

Executes the command of loading values.

This function executes the command of loading values.

**Parameters**

<i>in</i>	<i>instance</i>	PDB instance ID.
-----------	-----------------	------------------

Definition at line 293 of file pdb\_driver.c.

**16.101.4.15** void PDB\_DRV\_SetAdcPreTriggerDelayValue ( const uint32\_t *instance*, const uint32\_t *chn*, const uint32\_t *preChn*, const uint32\_t *value* )

Sets the ADC pre\_trigger delay value in the PDB module.

This function sets 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.

Definition at line 433 of file pdb\_driver.c.

**16.101.4.16** void PDB\_DRV\_SetCmpPulseOutDelayForHigh ( const uint32\_t *instance*, const uint32\_t *pulseChn*, const uint32\_t *value* )

Sets the CMP pulse out delay value for high in the PDB module.

This function sets the CMP pulse out delay value for high in the PDB module.

**Parameters**

<i>in</i>	<i>instance</i>	PDB instance ID.
<i>in</i>	<i>pulseChn</i>	Pulse channel.
<i>in</i>	<i>value</i>	Setting value.

Definition at line 471 of file pdb\_driver.c.

**16.101.4.17** void PDB\_DRV\_SetCmpPulseOutDelayForLow ( const uint32\_t *instance*, const uint32\_t *pulseChn*, const uint32\_t *value* )

Sets the CMP pulse out delay value for low in the PDB module.

This function sets the CMP pulse out delay value for low in the PDB module.

**Parameters**

<i>in</i>	<i>instance</i>	PDB instance ID.
<i>in</i>	<i>pulseChn</i>	Pulse channel.
<i>in</i>	<i>value</i>	Setting value.

Definition at line 489 of file pdb\_driver.c.

**16.101.4.18** void PDB\_DRV\_SetCmpPulseOutEnable ( const uint32\_t *instance*, const uint32\_t *pulseChnMask*, bool *enable* )

Switches on/off the CMP pulse out in the PDB module.

This function switches the CMP pulse on/off in the PDB module.

**Parameters**

<i>in</i>	<i>instance</i>	PDB instance ID.
<i>in</i>	<i>pulseChnMask</i>	Pulse channel mask.

<i>in</i>	<i>enable</i>	Switcher to assert the feature.
-----------	---------------	---------------------------------

Definition at line 454 of file pdb\_driver.c.

**16.101.4.19** void PDB\_DRV\_SetTimerModulusValue ( const uint32\_t *instance*, const uint16\_t *value* )

Sets the value of timer modulus.

This function sets the value of timer modulus.

**Parameters**

<i>in</i>	<i>instance</i>	PDB instance ID.
<i>in</i>	<i>value</i>	Setting value.

Definition at line 308 of file pdb\_driver.c.

**16.101.4.20** void PDB\_DRV\_SetValueForTimerInterrupt ( const uint32\_t *instance*, const uint16\_t *value* )

Sets the value for the timer interrupt.

This function sets the value for the timer interrupt.

**Parameters**

<i>in</i>	<i>instance</i>	PDB instance ID.
<i>in</i>	<i>value</i>	Setting value.

Definition at line 324 of file pdb\_driver.c.

**16.101.4.21** void PDB\_DRV\_SoftTriggerCmd ( const uint32\_t *instance* )

Triggers the PDB with a software trigger.

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.

**Parameters**

<i>in</i>	<i>instance</i>	PDB instance ID.
-----------	-----------------	------------------

Definition at line 232 of file pdb\_driver.c.

## 16.102 PINS Driver

### 16.102.1 Detailed Description

This section describes the programming interface of the PINS driver.

#### Data Structures

- struct [pin\\_settings\\_config\\_t](#)  
*Defines the converter configuration. [More...](#)*

#### Typedefs

- typedef uint8\_t [pins\\_level\\_type\\_t](#)  
*Type of a port levels representation. Implements : [pins\\_level\\_type\\_t\\_Class](#).*

#### Enumerations

- enum [port\\_data\\_direction\\_t](#) { [GPIO\\_INPUT\\_DIRECTION](#) = 0x0U, [GPIO\\_OUTPUT\\_DIRECTION](#) = 0x1U, [GPIO\\_UNSPECIFIED\\_DIRECTION](#) = 0x2U }  
*Configures the port data direction Implements : [port\\_data\\_direction\\_t\\_Class](#).*

#### PINS DRIVER API.

- status\_t [PINS\\_DRV\\_Init](#) (uint32\_t pinCount, const [pin\\_settings\\_config\\_t](#) config[ ])
  - Initializes the pins with the given configuration structure.*
- void [PINS\\_DRV\\_WritePin](#) (GPIO\_Type \*const base, pins\_channel\_type\_t pin, [pins\\_level\\_type\\_t](#) value)
  - Write a pin of a port with a given value.*
- void [PINS\\_DRV\\_WritePins](#) (GPIO\_Type \*const base, pins\_channel\_type\_t pins)
  - Write all pins of a port.*
- pins\_channel\_type\_t [PINS\\_DRV\\_GetPinsOutput](#) (const GPIO\_Type \*const base)
  - Get the current output from a port.*
- void [PINS\\_DRV\\_SetPins](#) (GPIO\_Type \*const base, pins\_channel\_type\_t pins)
  - Write pins with 'Set' value.*
- void [PINS\\_DRV\\_ClearPins](#) (GPIO\_Type \*const base, pins\_channel\_type\_t pins)
  - Write pins to 'Clear' value.*
- void [PINS\\_DRV\\_TogglePins](#) (GPIO\_Type \*const base, pins\_channel\_type\_t pins)
  - Toggle pins value.*
- pins\_channel\_type\_t [PINS\\_DRV\\_ReadPins](#) (const GPIO\_Type \*const base)
  - Read input pins.*

### 16.102.2 Data Structure Documentation

#### 16.102.2.1 struct pin\_settings\_config\_t

Defines the converter configuration.

This structure is used to configure the pins Implements : [pin\\_settings\\_config\\_t\\_Class](#)

Definition at line 565 of file pins\_driver.h.

## Data Fields

- uint32\_t [pinPortIdx](#)
- port\_mux\_t [mux](#)  
*Pin (C55: Out) mux selection.*
- GPIO\_Type \* [gpioBase](#)
- [port\\_data\\_direction\\_t](#) [direction](#)
- [pins\\_level\\_type\\_t](#) [initValue](#)

## Field Documentation

### 16.102.2.1.1 [port\\_data\\_direction\\_t](#) [direction](#)

Configures the port data direction.

Definition at line 602 of file pins\_driver.h.

### 16.102.2.1.2 [GPIO\\_Type\\*](#) [gpioBase](#)

GPIO base pointer.

Definition at line 601 of file pins\_driver.h.

### 16.102.2.1.3 [pins\\_level\\_type\\_t](#) [initValue](#)

Initial value

Definition at line 638 of file pins\_driver.h.

### 16.102.2.1.4 [port\\_mux\\_t](#) [mux](#)

Pin (C55: Out) mux selection.

Definition at line 588 of file pins\_driver.h.

### 16.102.2.1.5 [uint32\\_t](#) [pinPortIdx](#)

Port pin number.

Definition at line 572 of file pins\_driver.h.

## 16.102.3 Typedef Documentation

### 16.102.3.1 [typedef uint8\\_t](#) [pins\\_level\\_type\\_t](#)

Type of a port levels representation. Implements : [pins\\_level\\_type\\_t\\_Class](#).

Definition at line 53 of file pins\_driver.h.

## 16.102.4 Enumeration Type Documentation

### 16.102.4.1 [enum](#) [port\\_data\\_direction\\_t](#)

Configures the port data direction Implements : [port\\_data\\_direction\\_t\\_Class](#).

## Enumerator

**[GPIO\\_INPUT\\_DIRECTION](#)** General purpose input direction.

**[GPIO\\_OUTPUT\\_DIRECTION](#)** General purpose output direction.

**[GPIO\\_UNSPECIFIED\\_DIRECTION](#)** General purpose unspecified direction.

Definition at line 59 of file pins\_driver.h.



## 16.102.5 Function Documentation

16.102.5.1 void PINS\_DRV\_ClearPins ( GPIO\_Type \*const *base*, pins\_channel\_type\_t *pins* )

Write pins to 'Clear' value.

This function configures output pins listed in parameter *pins* (bits that are '1') to have a 'cleared' value (LOW). Pins corresponding to '0' will be unaffected.

## Parameters

in	<i>base</i>	GPIO base pointer (PTA, PTB, PTC, etc.)
in	<i>pins</i>	Pin mask of bits to be cleared. Each bit represents one pin (LSB is pin 0, MSB is pin 31). For each bit: <ul style="list-style-type: none"> <li>• 0: corresponding pin is unaffected</li> <li>• 1: corresponding pin is cleared(set to LOW)</li> </ul>

Definition at line 526 of file pins\_driver.c.

16.102.5.2 pins\_channel\_type\_t PINS\_DRV\_GetPinsOutput ( const GPIO\_Type \*const *base* )

Get the current output from a port.

This function returns the current output that is written to a port. Only pins that are configured as output will have meaningful values.

## Parameters

in	<i>base</i>	GPIO base pointer (PTA, PTB, PTC, etc.)
----	-------------	---

## Returns

GPIO outputs. Each bit represents one pin (LSB is pin 0, MSB is pin 31). For each bit:

- 0: corresponding pin is set to LOW
- 1: corresponding pin is set to HIGH

Definition at line 497 of file pins\_driver.c.

16.102.5.3 status\_t PINS\_DRV\_Init ( uint32\_t *pinCount*, const pin\_settings\_config\_t *config*[ ] )

Initializes the pins with the given configuration structure.

This function configures the pins with the options provided in the provided structure.

## Parameters

in	<i>pinCount</i>	The number of configured pins in structure
in	<i>config</i>	The configuration structure

## Returns

The status of the operation

Definition at line 50 of file pins\_driver.c.

16.102.5.4 pins\_channel\_type\_t PINS\_DRV\_ReadPins ( const GPIO\_Type \*const *base* )

Read input pins.

This function returns the current input values from a port. Only pins configured as input will have meaningful values.

**Parameters**

<i>in</i>	<i>base</i>	GPIO base pointer (PTA, PTB, PTC, etc.)
-----------	-------------	---

**Returns**

GPIO inputs. Each bit represents one pin (LSB is pin 0, MSB is pin 31). For each bit:

- 0: corresponding pin is read as LOW
- 1: corresponding pin is read as HIGH

Definition at line 554 of file pins\_driver.c.

**16.102.5.5** void PINS\_DRV\_SetPins ( GPIO\_Type \*const *base*, pins\_channel\_type\_t *pins* )

Write pins with 'Set' value.

This function configures output pins listed in parameter *pins* (bits that are '1') to have a value of 'set' (HIGH). Pins corresponding to '0' will be unaffected.

**Parameters**

<i>in</i>	<i>base</i>	GPIO base pointer (PTA, PTB, PTC, etc.)
<i>in</i>	<i>pins</i>	Pin mask of bits to be set. Each bit represents one pin (LSB is pin 0, MSB is pin 31). For each bit: <ul style="list-style-type: none"> <li>• 0: corresponding pin is unaffected</li> <li>• 1: corresponding pin is set to HIGH</li> </ul>

Definition at line 511 of file pins\_driver.c.

**16.102.5.6** void PINS\_DRV\_TogglePins ( GPIO\_Type \*const *base*, pins\_channel\_type\_t *pins* )

Toggle pins value.

This function toggles output pins listed in parameter *pins* (bits that are '1'). Pins corresponding to '0' will be unaffected.

**Parameters**

<i>in</i>	<i>base</i>	GPIO base pointer (PTA, PTB, PTC, etc.)
<i>in</i>	<i>pins</i>	Pin mask of bits to be toggled. Each bit represents one pin (LSB is pin 0, MSB is pin 31). For each bit: <ul style="list-style-type: none"> <li>• 0: corresponding pin is unaffected</li> <li>• 1: corresponding pin is toggled</li> </ul>

Definition at line 540 of file pins\_driver.c.

**16.102.5.7** void PINS\_DRV\_WritePin ( GPIO\_Type \*const *base*, pins\_channel\_type\_t *pin*, pins\_level\_type\_t *value* )

Write a pin of a port with a given value.

This function writes the given pin from a port, with the given value ('0' represents LOW, '1' represents HIGH).

**Parameters**

<i>in</i>	<i>base</i>	GPIO base pointer (PTA, PTB, PTC, etc.)
-----------	-------------	---

in	<i>pin</i>	Pin number to be written
in	<i>value</i>	Pin value to be written <ul style="list-style-type: none"><li>• 0: corresponding pin is set to LOW</li><li>• 1: corresponding pin is set to HIGH</li></ul>

Definition at line 468 of file pins\_driver.c.

**16.102.5.8** void PINS\_DRV\_WritePins ( GPIO\_Type \*const *base*, pins\_channel\_type\_t *pins* )

Write all pins of a port.

This function writes all pins configured as output with the values given in the parameter pins. '0' represents LOW, '1' represents HIGH.

**Parameters**

in	<i>base</i>	GPIO base pointer (PTA, PTB, PTC, etc.)
in	<i>pins</i>	Pin mask to be written <ul style="list-style-type: none"><li>• 0: corresponding pin is set to LOW</li><li>• 1: corresponding pin is set to HIGH</li></ul>

Definition at line 483 of file pins\_driver.c.

### 16.103 Peripheral access layer for S32K142

This module covers all memory mapped register available on SoC.

## 16.104 Pins Driver (PINS)

### 16.104.1 Detailed Description

The S32 SDK provides Peripheral Drivers for the PINS module of S32K1xx, S32MTV, S32V234, MPC574xx and S32Rx7x devices.

The module provides dedicated pad control to general-purpose pads that can be configured as either inputs or outputs. The PINS module provides registers that enable user software to read values from GPIO pads configured as inputs, and write values to GPIO pads configured as outputs:

- When configured as output, you can write to an internal register to control the state driven on the associated output pad.
- When configured as input, you can detect the state of the associated pad by reading the value from an internal register.
- When configured as input and output, the pad value can be read back, which can be used as a method of checking if the written value appeared on the pad.

The PINS supports these following features: For S32K1xx and S32MTV devices: Pins driver is based on PORT (Port Control and Interrupt) and GPIO (General-Purpose Input/Output) modules Pin interrupt

- Interrupt flag and enable registers for each pin
- Support for edge sensitive (rising, falling, both) or level sensitive (low, high) configured per pin
- Support for interrupt or DMA request configured per pin
- Asynchronous wake-up in low-power modes
- Pin interrupt is functional in all digital pin muxing modes
- Peripheral trigger output (active high, low) configured per pin Digital input filter
- Digital input filter for each pin, usable by any digital peripheral muxed onto the pin
- Individual enable or bypass control field per pin
- Selectable clock source for digital input filter with a five bit resolution on filter size
- Functional in all digital pin multiplexing modes Port control
- Individual pull control fields with pullup, pulldown, and pull-disable support
- Individual drive strength field supporting high and low drive strength
- Individual slew rate field supporting fast and slow slew rates
- Individual input passive filter field supporting enable and disable of the individual input passive filter
- Individual open drain field supporting enable and disable of the individual open drain output
- Individual over-current detect enable with over-current detect flag and associated interrupt
- Individual mux control field supporting analog or pin disabled, GPIO, and up to 6 chip-specific digital functions
- Pad configuration fields are functional in all digital pin muxing modes For S32V234, MPC574xx and S32Rx7x devices: Pins driver is based on SIUL2 (System Integration Unit Lite2) module The System Integration Unit Lite2 supports these distinctive features:
  - 1 to 32 GPIO ports with data control
  - Drive data to as many as 16 independent I/O channels
  - Sample data from as many as 16 independent I/O channels Two 16-bit registers can be read/written with one access for a 32-bit port, if needed. External interrupt/DMA request support with:

- 1 to 4 system interrupt vectors for 1 to 4 interrupt sources with independent interrupt masks. For 32 external interrupt sources (REQ pins), four groups have eight interrupt sources each, and each of the four groups is assigned one system interrupt vector.
- 1 to 32 programmable digital glitch filters, one for each REQ pin
- 1 to 4 system DMA request channels up to 32 REQ pins, depending on device using
- Edge detection Additionally the SIUL2 contains the Multiplexed Signal Configuration Registers (MSCR) that configure the electrical parameters and settings for as many as 512 functional pads. The number of these registers that is actually implemented varies by device. These registers configure the following pad features:
  - Drive strength
  - Output impedance control
  - Open drain/source output enable
  - Slew rate control
  - Hysteresis control
  - Inversion control
  - Internal pull control and pull selection
  - Pin function assignment
  - Control of analog path switches
  - Safe mode behavior configuration

#### Modules

- [PINS Driver](#)

## 16.105 Power Manager

### 16.105.1 Detailed Description

The S32 SDK Power Manager provides a set of API/services that enables applications to configure and select among various operational and low power modes.

#### Driver consideration

The Power Manager driver is developed on top of an appropriate hardware access layer. The 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. The Power Manager provides a notification mechanism for registered callbacks and API for static and dynamic callback registration.

The Driver uses structures for configuration. The actual format of the structure is defined by the underlying device specific header file. There is a power mode and a callback configuration structure. These structures may be generated using Processor Expert. The user application can use the default for most settings, changing only what is necessary.

This driver provides functions for initializing power manager and changing the power mode.

All methods that access the hardware layer will return an error code to signal if the operation succeeded or failed. The values are defined by the `status_t` enumeration, and the possible values include: success, switch error, callback notification errors, wrong clock setup error.

#### Modules

- [Power Manager Driver](#)

*This module covers the device-specific power\_manager functionality implemented for S32K1xx, s32k14xW and S32MTV SOC.*

- [Power\\_s32k1xx](#)

#### Data Structures

- struct [power\\_manager\\_notify\\_struct\\_t](#)  
*Power mode user configuration structure. [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 status\_t(\* [power\\_manager\\_callback\\_t](#)) ([power\\_manager\\_notify\\_struct\\_t](#) \*notify, [power\\_manager\\_callback\\_data\\_t](#) \*dataPtr)  
*Callback prototype.*

#### Enumerations

- enum [power\\_manager\\_policy\\_t](#) { [POWER\\_MANAGER\\_POLICY\\_AGREEMENT](#), [POWER\\_MANAGER\\_POLICY\\_FORCIBLE](#) }  
*Power manager policies.*
- enum [power\\_manager\\_notify\\_t](#) { [POWER\\_MANAGER\\_NOTIFY\\_RECOVER](#) = 0x00U, [POWER\\_MANAGER\\_NOTIFY\\_BEFORE](#) = 0x01U, [POWER\\_MANAGER\\_NOTIFY\\_AFTER](#) = 0x02U }

The PM notification type. Used to notify registered callbacks. Callback notifications can be invoked in following situations:

- enum `power_manager_callback_type_t` { `POWER_MANAGER_CALLBACK_BEFORE` = 0x01U, `POWER_MANAGER_CALLBACK_AFTER` = 0x02U, `POWER_MANAGER_CALLBACK_BEFORE_AFTER` = 0x03U }

The callback type indicates when a callback will be invoked.

## Functions

- status\_t `POWER_SYS_Init` (`power_manager_user_config_t` \*(\*powerConfigsPtr)[ ], uint8\_t configsNumber, `power_manager_callback_user_config_t` \*(\*callbacksPtr)[ ], uint8\_t callbacksNumber)  
Power manager initialization for operation.
- status\_t `POWER_SYS_Deinit` (void)  
This function deinitializes the Power manager.
- status\_t `POWER_SYS_SetMode` (uint8\_t powerModelIndex, `power_manager_policy_t` policy)  
This function configures the power mode.
- status\_t `POWER_SYS_GetLastMode` (uint8\_t \*powerModelIndexPtr)  
This function returns the last successfully set power mode.
- status\_t `POWER_SYS_GetLastModeConfig` (`power_manager_user_config_t` \*\*powerModePtr)  
This function returns the user configuration structure of the last successfully set power mode.
- `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 callback configuration structure for the last failed notification.
- void `POWER_SYS_GetDefaultConfig` (`power_manager_user_config_t` \*const config)  
This function returns the default power\_manager configuration structure.

## Variables

- `power_manager_state_t` `gPowerManagerState`  
Power manager internal structure.

### 16.105.2 Data Structure Documentation

#### 16.105.2.1 struct power\_manager\_notify\_struct\_t

Power mode user configuration structure.

This structure defines power mode with additional power options. This structure is implementation-defiend. Please refer to actual definition based on the underlying HAL (SMC, MC\_ME etc). Applications may define multiple power modes and switch between them. A list of all 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 takes as argument the index of the requested power mode in the list passed during manager initialization. The power mode currently in use can be retrieved by calling `POWER_SYS_GetLastMode()`, which provides the index of the current power mode, or by calling `POWER_SYS_GetLastModeConfig()`, which provides a pointer to the configuration structure of the current power mode. The members of the power mode configuration structure depend on power options available for a specific chip, and includes at least the power mode. The available power modes are chip-specific. See `power_manager_modes_t` defined in the underlying HAL for a list of all supported modes.

Power notification structure passed to registered callback function

Implements `power_manager_notify_struct_t_Class`

Definition at line 140 of file `power_manager.h`.



## Data Fields

- [power\\_manager\\_user\\_config\\_t](#) \* targetPowerConfigPtr
- [uint8\\_t](#) targetPowerConfigIndex
- [power\\_manager\\_policy\\_t](#) policy
- [power\\_manager\\_notify\\_t](#) notifyType

## Field Documentation

16.105.2.1.1 [power\\_manager\\_notify\\_t](#) notifyType

Power mode notification type.

Definition at line 145 of file power\_manager.h.

16.105.2.1.2 [power\\_manager\\_policy\\_t](#) policy

Power mode transition policy.

Definition at line 144 of file power\_manager.h.

16.105.2.1.3 [uint8\\_t](#) targetPowerConfigIndex

Target power configuration index.

Definition at line 143 of file power\_manager.h.

16.105.2.1.4 [power\\_manager\\_user\\_config\\_t](#)\* targetPowerConfigPtr

Pointer to target power configuration

Definition at line 142 of file power\_manager.h.

16.105.2.2 [struct power\\_manager\\_callback\\_user\\_config\\_t](#)

callback configuration structure

This structure holds configuration of callbacks passed to the Power manager during its initialization. Structures 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 Implements [power\\_manager\\_callback\\_user\\_config\\_t](#) Class

Definition at line 185 of file power\_manager.h.

## Data Fields

- [power\\_manager\\_callback\\_t](#) callbackFunction
- [power\\_manager\\_callback\\_type\\_t](#) callbackType
- [power\\_manager\\_callback\\_data\\_t](#) \* callbackData

## Field Documentation

16.105.2.2.1 [power\\_manager\\_callback\\_data\\_t](#)\* callbackData

Definition at line 189 of file power\_manager.h.

16.105.2.2.2 [power\\_manager\\_callback\\_t](#) callbackFunction

Definition at line 187 of file power\_manager.h.

16.105.2.2.3 [power\\_manager\\_callback\\_type\\_t](#) callbackType

Definition at line 188 of file power\_manager.h.

### 16.105.2.3 struct power\_manager\_state\_t

Power manager internal state structure.

Power manager internal structure. Contains data necessary for Power manager proper functionality. Stores references to registered power mode configurations, callbacks, and other internal data. This structure is statically allocated and initialized by `POWER_SYS_Init()`. Implements `power_manager_state_t_Class`

Definition at line 201 of file `power_manager.h`.

#### Data Fields

- `power_manager_user_config_t` `*( * configs )[]`
- `uint8_t configsNumber`
- `power_manager_callback_user_config_t` `*( * staticCallbacks )[]`
- `uint8_t staticCallbacksNumber`
- `uint8_t errorCallbackIndex`
- `uint8_t currentConfig`

#### Field Documentation

##### 16.105.2.3.1 power\_manager\_user\_config\_t\*( \* configs )[]

Pointer to power configure table.

Definition at line 203 of file `power_manager.h`.

##### 16.105.2.3.2 uint8\_t configsNumber

Number of power configurations

Definition at line 204 of file `power_manager.h`.

##### 16.105.2.3.3 uint8\_t currentConfig

Index of current configuration.

Definition at line 208 of file `power_manager.h`.

##### 16.105.2.3.4 uint8\_t errorCallbackIndex

Index of callback returns error.

Definition at line 207 of file `power_manager.h`.

##### 16.105.2.3.5 power\_manager\_callback\_user\_config\_t\*( \* staticCallbacks )[]

Pointer to callback table.

Definition at line 205 of file `power_manager.h`.

##### 16.105.2.3.6 uint8\_t staticCallbacksNumber

Max. number of callback configurations

Definition at line 206 of file `power_manager.h`.

### 16.105.3 Typedef Documentation

#### 16.105.3.1 typedef void power\_manager\_callback\_data\_t

Callback-specific data.

Pointer to data of this type is passed during callback registration. The pointer is part of the [power\\_manager\\_callback\\_user\\_config\\_t](#) structure and is passed to the callback during power mode change notifications. Implements `power_manager_callback_data_t_Class`

Definition at line 115 of file `power_manager.h`.

```
16.105.3.2 typedef status_t(* power_manager_callback_t) (power_manager_notify_struct_t *notify,
power_manager_callback_data_t *dataPtr)
```

Callback prototype.

Declaration of callback. It is common for all registered callbacks. Function pointer of this type is part of [power\\_manager\\_callback\\_user\\_config\\_t](#) callback configuration structure. Depending on the callback type, the callback function is invoked during power mode change (see [POWER\\_SYS\\_SetMode\(\)](#)) before the mode change, after it, or in both cases to notify about the change progress (see `power_manager_callback_type_t`). When called, the type of the notification is passed as parameter along with a pointer to 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 a 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 other than `STATUS_SUCCESS` (see [POWER\\_SYS\\_SetMode\(\)](#)).

Parameters

<i>notify</i>	Notification structure.
<i>dataPtr</i>	Callback data. Pointer 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 `STATUS_SUCCESS`. Implements `power_manager_callback_t_Class`

Definition at line 169 of file `power_manager.h`.

#### 16.105.4 Enumeration Type Documentation

##### 16.105.4.1 enum power\_manager\_callback\_type\_t

The callback type indicates when a callback will be invoked.

Used in the callback configuration structures ([power\\_manager\\_callback\\_user\\_config\\_t](#)) to specify when the registered callback will be called during power mode change initiated by [POWER\\_SYS\\_SetMode\(\)](#).

Implements `power_manager_callback_type_t_Class`

Enumerator

**`POWER_MANAGER_CALLBACK_BEFORE`** Before callback.  
**`POWER_MANAGER_CALLBACK_AFTER`** After callback.  
**`POWER_MANAGER_CALLBACK_BEFORE_AFTER`** Before-After callback.

Definition at line 100 of file `power_manager.h`.

##### 16.105.4.2 enum power\_manager\_notify\_t

The PM notification type. Used to notify registered callbacks. Callback notifications can be invoked in following situations:

- before a 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 a successful change of the power mode.

- after an unsuccessful attempt to switch power mode, in order to recover to a working state. Implements `power_manager_notify_t_Class`

#### Enumerator

**`POWER_MANAGER_NOTIFY_RECOVER`** Notify IP to recover to previous work state.

**`POWER_MANAGER_NOTIFY_BEFORE`** Notify IP that the system will change the power setting.

**`POWER_MANAGER_NOTIFY_AFTER`** Notify IP that the system has changed to a new power setting.

Definition at line 84 of file `power_manager.h`.

#### 16.105.4.3 `enum power_manager_policy_t`

Power manager policies.

Defines whether the mode switch initiated by the [POWER\\_SYS\\_SetMode\(\)](#) is agreed upon (depending on the result of notification callbacks), or forced. For `POWER_MANAGER_POLICY_FORCIBLE` the power mode is changed regardless of the callback results, while for `POWER_MANAGER_POLICY_AGREEMENT` policy any error code returned by one of the callbacks aborts the mode change. See also [POWER\\_SYS\\_SetMode\(\)](#) description. Implements `power_manager_policy_t_Class`

#### Enumerator

**`POWER_MANAGER_POLICY_AGREEMENT`** Power mode is changed if all of the callbacks return success.

**`POWER_MANAGER_POLICY_FORCIBLE`** Power mode is changed regardless of the result of callbacks.

Definition at line 69 of file `power_manager.h`.

#### 16.105.5 Function Documentation

##### 16.105.5.1 `status_t POWER_SYS_Deinit ( void )`

This function deinitializes the Power manager.

#### Returns

An error code or `STATUS_SUCCESS`.

Definition at line 110 of file `power_manager.c`.

##### 16.105.5.2 `power_manager_modes_t POWER_SYS_GetCurrentMode ( void )`

This function returns currently running power mode.

This function reads hardware settings and returns currently running power mode.

#### Returns

Currently used run power mode.

Definition at line 244 of file `power_manager_S32K1xx.c`.

##### 16.105.5.3 `void POWER_SYS_GetDefaultConfig ( power_manager_user_config_t *const config )`

This function returns the default `power_manager` configuration structure.

This function returns a pointer of the `power_manager` configuration structure. All structure members have default value when CPU is default power mode.

Definition at line 406 of file `power_manager.c`.

#### 16.105.5.4 `power_manager_callback_user_config_t*` `POWER_SYS_GetErrorCallback ( void )`

This function returns the callback configuration structure for the last failed notification.

This function returns a pointer to configuration structure of the last callback that failed during the power mode switch when `POWER_SYS_SetMode()` was called. If the last `POWER_SYS_SetMode()` call ended successfully, a NULL value is returned.

##### Returns

Pointer to the callback configuration which returns error.

Definition at line 208 of file `power_manager.c`.

#### 16.105.5.5 `uint8_t` `POWER_SYS_GetErrorCallbackIndex ( void )`

This function returns the last failed notification callback.

This function returns the index of the last callback that failed during the power mode switch when `POWER_SYS_SetMode()` was called. The returned value represents the index in the array of registered callbacks. If the last `POWER_SYS_SetMode()` call ended successfully, a value equal to the number of registered callbacks is returned.

##### Returns

Callback index of last failed callback or value equal to callbacks count.

Definition at line 196 of file `power_manager.c`.

#### 16.105.5.6 `status_t` `POWER_SYS_GetLastMode ( uint8_t * powerModeIndexPtr )`

This function returns the last successfully set power mode.

This function returns index of power mode which was last set using `POWER_SYS_SetMode()`. If the power mode was entered even though some of the registered callbacks 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 `STATUS_ERROR` value.

##### Parameters

out	<i>powerModeIndexPtr</i>	Power mode which has been set represented as an index into array of power mode configurations passed to the <code>POWER_SYS_Init()</code> .
-----	--------------------------	---

##### Returns

An error code or `STATUS_SUCCESS`.

Definition at line 133 of file `power_manager.c`.

#### 16.105.5.7 `status_t` `POWER_SYS_GetLastModeConfig ( power_manager_user_config_t ** powerModePtr )`

This function returns the user configuration structure of the last successfully set power mode.

This function returns a pointer to configuration structure which was last set using `POWER_SYS_SetMode()`. If the current power mode was entered even though some of the registered callbacks 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 `STATUS_ERROR` value.

##### Parameters

out	<i>powerModePtr</i>	Pointer to power mode configuration structure of the last set power mode.
-----	---------------------	---

#### Returns

An error code or STATUS\_SUCCESS.

Definition at line 165 of file power\_manager.c.

**16.105.5.8** `status_t POWER_SYS_Init ( power_manager_user_config_t (*)(*) powerConfigsPtr[], uint8_t configsNumber, power_manager_callback_user_config_t (*)(*) callbacksPtr[], uint8_t callbacksNumber )`

Power manager initialization for operation.

This function initializes the Power manager and its run-time state structure. Pointer to an array of Power mode configuration structures needs to be passed as a parameter along with a parameter specifying its size. At least one power mode configuration is required. Optionally, pointer to the array of predefined callbacks can be passed with its corresponding size parameter. For details about callbacks, refer to the [power\\_manager\\_callback\\_user\\_config\\_t](#). As Power manager stores only pointers to arrays of these structures, they need to exist and be valid for the entire life cycle of Power manager.

#### Parameters

in	<i>powerConfigsPtr</i>	A pointer to an array of pointers to all power configurations which will be handled by Power manager.
in	<i>configsNumber</i>	Number of power configurations. Size of powerConfigsPtr array.
in	<i>callbacksPtr</i>	A pointer to an array of pointers to callback configurations. If there are no callbacks to register during Power manager initialization, use NULL value.
in	<i>callbacksNumber</i>	Number of registered callbacks. Size of callbacksPtr array.

#### Returns

An error code or STATUS\_SUCCESS.

Definition at line 70 of file power\_manager.c.

**16.105.5.9** `status_t POWER_SYS_SetMode ( uint8_t powerModelIndex, power_manager_policy_t policy )`

This function configures the power mode.

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 POWER\_MANAGER\_CALLBACK\_BEFORE 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 POWER\_MANAGER\_CALLBACK\_AFTER). 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(POWER\_MANAGER\_POLICY\_FORCIBLE) or aborted(POWER\_MANAGER\_POLICY\_AGREEMENT). When mode change is forced, the results of the before switch notifications are ignored. If agreement is requested, in case any callback returns an error code then further before switch notifications are cancelled and all already notified callbacks are re-invoked with POWER\_MANAGER\_CALLBACK\_AFTER set as callback type parameter. The index of the callback which returned error code during pre-switch notifications is stored and can be obtained by using [POWER\\_SYS\\_GetErrorCallback\(\)](#). Any error codes during callbacks re-invocation (recover phase) are ignored. [POWER\\_SYS\\_SetMode\(\)](#) returns an error code denoting the phase in which a callback failed. It is possible to enter any mode supported by the processor. Refer to the chip reference manual for the list of available power modes. If it is necessary to switch into an intermediate power mode prior to entering the 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**

in	<i>powerMode↔ Index</i>	Requested power mode represented as an index into array of user-defined power mode configurations passed to the <a href="#">POWER_SYS_Init()</a> .
in	<i>policy</i>	Transaction policy

**Returns**

An error code or STATUS\_SUCCESS.

Definition at line 323 of file power\_manager.c.

**16.105.6 Variable Documentation****16.105.6.1 power\_manager\_state\_t gPowerManagerState**

Power manager internal structure.

Definition at line 52 of file power\_manager\_S32K1xx.c.



## 16.106 Power Manager Driver

This module covers the device-specific power\_manager functionality implemented for S32K1xx, s32k14xW and S32MTV SOC.

### Hardware background

System mode controller (SMC) is passing the system into and out of all low-power Stop and Run modes. Controls the power, clocks and memories of the system to achieve the power consumption and functionality of that mode.

### Driver consideration

Power mode entry and sleep-on-exit-value are provided at initialization time through the power manager user configuration structure.

With platform is S32K14x, the available power mode entries are the following ones: HSRUN, RUN, VLPR, STOP1, STOP2 and VLPS.

With platform is S32MTV,S32K11x and S32K14xW. The available power mode entries are the following ones: RUN, VLPR, STOP1, STOP2 and VLPS.

This is an example of configuration:

```
power_manager_user_config_t pwrMan1_InitConfig0 = {
    .powerMode = POWER_MANAGER_RUN,
    .sleepOnExitValue = false,
};

power_manager_user_config_t *powerConfigsArr[] = {
    &pwrMan1_InitConfig0
};

power_manager_callback_user_config_t * powerCallbacksConfigsArr[] = {(
    void *)0};

if (STATUS_SUCCESS != POWER_SYS_Init(&powerConfigsArr,1,&powerCallbacksConfigsArr,0)) {
    ...
}
else {
    ...
}

if (STATUS_SUCCESS != POWER_SYS_SetMode(0,
    POWER_MANAGER_POLICY_AGREEMENT)) {
    ...
}
else {
    ...
}
```

### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\drivers\src\power_manager.c
${S32SDK_PATH}\platform\drivers\src\S32K1xx\power_manager_S32K1xx.c
${S32SDK_PATH}\platform\drivers\src\S32K1xx\power_smc_hw_access.c
```

### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc\
${S32SDK_PATH}\platform\drivers\src\power\
${S32SDK_PATH}\platform\drivers\src\power\S32K1xx\
```

### Compile symbols

No special symbols are required for this component

## Dependencies

## [Clock Manager](#)

### Important Note

1. ERR01077: The SCG\_RCCR[SCS] and SCG\_HCCR[SCS] may have a corrupted status during the interval by the software to ensure when the system clock is switching.  
This errata did workaround by the SCS field was read twice the system clock switch has completed.  
The clock configuration is not immediately updated after MCU switched from very low power mode to run or high speed mode. It may be taking longer time than expected.
2. The power manager driver will disable SPLL, FIRC, SOSC source in RUN mode before MCU jumps from RUN,HSRUN to very low power mode.  
SIRC is clock source when MCU enters very low power mode. Driver will update initialize system clock configuration when MCU jumps to RUN or HSRUN mode again.  
It will enable all clock source again which is configured by clock configurations. This is executed when user calls the POWER\_SYS\_SetMode function for RUN or HSRUN.
3. When MCU switches from HSRUN to STOP or VLP mode, the driver code will auto switch RUN mode before MCU enters next mode.
4. Users need to take care peripherals clock frequency via SPLL DIVx\_CLK, SIRC DIVx\_CLK after MCU switched power mode.  
Clock configuration can be re-initialized when MCU returns to RUN mode. This way will make sure the clock for all peripherals.

## 16.107 Power\_s32k1xx

### 16.107.1 Detailed Description

#### Data Structures

- struct [power\\_manager\\_user\\_config\\_t](#)  
*Power mode user configuration structure. [More...](#)*
- struct [smc\\_power\\_mode\\_protection\\_config\\_t](#)  
*Power mode protection configuration. [More...](#)*
- struct [smc\\_power\\_mode\\_config\\_t](#)  
*Power mode control configuration used for calling the SMC\_SYS\_SetPowerMode API. [More...](#)*

#### Enumerations

- enum [power\\_manager\\_modes\\_t](#) { [POWER\\_MANAGER\\_RUN](#), [POWER\\_MANAGER\\_VLPR](#), [POWER\\_MANAGER\\_VLPS](#), [POWER\\_MANAGER\\_MAX](#) }  
*Power modes enumeration.*
- enum [power\\_mode\\_stat\\_t](#) {  
[STAT\\_RUN](#) = 0x01, [STAT\\_STOP](#) = 0x02, [STAT\\_VLPR](#) = 0x04, [STAT\\_VLPW](#) = 0x08,  
[STAT\\_VLPS](#) = 0x10, [STAT\\_HSRUN](#) = 0x80, [STAT\\_INVALID](#) = 0xFF }  
*Power Modes in PMSTAT.*
- enum [smc\\_run\\_mode\\_t](#) { [SMC\\_RUN](#), [SMC\\_RESERVED\\_RUN](#), [SMC\\_VLPR](#), [SMC\\_HSRUN](#) }  
*Run mode definition.*
- enum [smc\\_stop\\_mode\\_t](#) { [SMC\\_STOP](#) = 0U, [SMC\\_RESERVED\\_STOP1](#) = 1U, [SMC\\_VLPS](#) = 2U }  
*Stop mode definition.*
- enum [smc\\_stop\\_option\\_t](#) { [SMC\\_STOP\\_RESERVED](#) = 0x00, [SMC\\_STOP1](#) = 0x01, [SMC\\_STOP2](#) = 0x02 }  
*STOP option.*
- enum [rcm\\_source\\_names\\_t](#) {  
[RCM\\_LOW\\_VOLT\\_DETECT](#) = 1U, [RCM\\_LOSS\\_OF\\_CLK](#) = 2U, [RCM\\_LOSS\\_OF\\_LOCK](#) = 3U, [RCM\\_WATCHDOG](#) = 5U,  
[RCM\\_EXTERNAL\\_PIN](#) = 6U, [RCM\\_POWER\\_ON](#) = 7U, [RCM\\_SJTAG](#) = 8U, [RCM\\_CORE\\_LOCKUP](#) = 9U,  
[RCM\\_SOFTWARE](#) = 10U, [RCM\\_SMDM\\_AP](#) = 11U, [RCM\\_STOP\\_MODE\\_ACK\\_ERR](#) = 13U, [RCM\\_SRC\\_NAME\\_MAX](#) }  
*System Reset Source Name definitions Implements rcm\_source\_names\_t\_Class.*

#### Functions

- status\_t [POWER\\_SYS\\_DoInit](#) (void)  
*This function implementation-specific configuration of power modes.*
- status\_t [POWER\\_SYS\\_DoDeinit](#) (void)  
*This function implementation-specific de-initialization of power manager.*
- status\_t [POWER\\_SYS\\_DoSetMode](#) (const [power\\_manager\\_user\\_config\\_t](#) \*const configPtr)  
*This function configures the power mode.*
- bool [POWER\\_SYS\\_GetResetSrcStatusCmd](#) (const RCM\_Type \*const baseAddr, const [rcm\\_source\\_names\\_t](#) srcName)  
*Gets the reset source status.*
- static void [POWER\\_SYS\\_DoGetDefaultConfig](#) ([power\\_manager\\_user\\_config\\_t](#) \*const defaultConfig)  
*Gets the default power\_manager configuration structure.*

## 16.107.2 Data Structure Documentation

### 16.107.2.1 struct power\_manager\_user\_config\_t

Power mode user configuration structure.

List of power mode configuration structure members depends on power options available for the specific chip. Complete list contains: mode - S32K power mode. List of available modes is chip-specific. See power\_manager\_modes\_t list of modes. sleepOnExitOption - Controls whether the sleep-on-exit option value is used (when set to true) or ignored (when set to false). See sleepOnExitValue. sleepOnExitValue - When set to true, ARM core returns to sleep (S32K wait modes) or deep sleep state (S32K stop modes) after interrupt service finishes. When set to false, core stays woken-up. Implements power\_manager\_user\_config\_t\_Class

Definition at line 95 of file power\_manager\_S32K1xx.h.

#### Data Fields

- [power\\_manager\\_modes\\_t powerMode](#)
- bool [sleepOnExitValue](#)

#### Field Documentation

##### 16.107.2.1.1 power\_manager\_modes\_t powerMode

Definition at line 97 of file power\_manager\_S32K1xx.h.

##### 16.107.2.1.2 bool sleepOnExitValue

Definition at line 98 of file power\_manager\_S32K1xx.h.

### 16.107.2.2 struct smc\_power\_mode\_protection\_config\_t

Power mode protection configuration.

Definition at line 153 of file power\_manager\_S32K1xx.h.

#### Data Fields

- bool [vlpProt](#)

#### Field Documentation

##### 16.107.2.2.1 bool vlpProt

VLP protect

Definition at line 155 of file power\_manager\_S32K1xx.h.

### 16.107.2.3 struct smc\_power\_mode\_config\_t

Power mode control configuration used for calling the SMC\_SYS\_SetPowerMode API.

Definition at line 165 of file power\_manager\_S32K1xx.h.

#### Data Fields

- [power\\_manager\\_modes\\_t powerModeName](#)

#### Field Documentation

##### 16.107.2.3.1 power\_manager\_modes\_t powerModeName

Power mode(enum), see power\_manager\_modes\_t

Definition at line 167 of file power\_manager\_S32K1xx.h.

## 16.107.3 Enumeration Type Documentation

## 16.107.3.1 enum power\_manager\_modes\_t

Power modes enumeration.

Defines power modes. Used in the power mode configuration structure ([power\\_manager\\_user\\_config\\_t](#)). From ARM core perspective, Power modes can be generally divided into 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:

- POWER\_MANAGER\_HSRUN - High speed run mode.
- POWER\_MANAGER\_RUN - Run mode.
- POWER\_MANAGER\_VLPR - Very low power run mode.
- POWER\_MANAGER\_WAIT - Wait mode.
- POWER\_MANAGER\_VLPW - Very low power wait mode.
- POWER\_MANAGER\_PSTOP1 - Partial stop 1 mode.
- POWER\_MANAGER\_PSTOP2 - Partial stop 2 mode.
- POWER\_MANAGER\_PSTOP1 - Stop 1 mode.
- POWER\_MANAGER\_PSTOP2 - Stop 2 mode.
- POWER\_MANAGER\_VLPS - Very low power stop mode. Implements power\_manager\_modes\_t\_Class

## Enumerator

**POWER\_MANAGER\_RUN** Run mode.  
**POWER\_MANAGER\_VLPR** Very low power run mode.  
**POWER\_MANAGER\_VLPS** Very low power stop mode.  
**POWER\_MANAGER\_MAX**

Definition at line 58 of file power\_manager\_S32K1xx.h.

## 16.107.3.2 enum power\_mode\_stat\_t

Power Modes in PMSTAT.

## Enumerator

**STAT\_RUN** 0000\_0001 - Current power mode is RUN  
**STAT\_STOP** 0000\_0010 - Current power mode is STOP  
**STAT\_VLPR** 0000\_0100 - Current power mode is VLPR  
**STAT\_VLPW** 0000\_1000 - Current power mode is VLPW  
**STAT\_VLPS** 0001\_0000 - Current power mode is VLPS  
**STAT\_HSRUN** 1000\_0000 - Current power mode is HSRUN  
**STAT\_INVALID** 1111\_1111 - Non-existing power mode

Definition at line 105 of file power\_manager\_S32K1xx.h.

16.107.3.3 enum `rcm_source_names_t`

System Reset Source Name definitions Implements `rcm_source_names_t` Class.

## Enumerator

***RCM\_LOW\_VOLT\_DETECT*** Low voltage detect reset  
***RCM\_LOSS\_OF\_CLK*** Loss of clock reset  
***RCM\_LOSS\_OF\_LOCK*** Loss of lock reset  
***RCM\_WATCH\_DOG*** Watch dog reset  
***RCM\_EXTERNAL\_PIN*** External pin reset  
***RCM\_POWER\_ON*** Power on reset  
***RCM\_SJTAG*** JTAG generated reset  
***RCM\_CORE\_LOCKUP*** core lockup reset  
***RCM\_SOFTWARE*** Software reset  
***RCM\_SMDM\_AP*** MDM-AP system reset  
***RCM\_STOP\_MODE\_ACK\_ERR*** Stop mode ack error reset  
***RCM\_SRC\_NAME\_MAX***

Definition at line 181 of file `power_manager_S32K1xx.h`.

16.107.3.4 enum `smc_run_mode_t`

Run mode definition.

## Enumerator

***SMC\_RUN*** normal RUN mode  
***SMC\_RESERVED\_RUN***  
***SMC\_VLPR*** Very-Low-Power RUN mode  
***SMC\_HSRUN*** High Speed Run mode (HSRUN)

Definition at line 120 of file `power_manager_S32K1xx.h`.

16.107.3.5 enum `smc_stop_mode_t`

Stop mode definition.

## Enumerator

***SMC\_STOP*** Normal STOP mode  
***SMC\_RESERVED\_STOP1*** Reserved  
***SMC\_VLPS*** Very-Low-Power STOP mode

Definition at line 131 of file `power_manager_S32K1xx.h`.

16.107.3.6 enum `smc_stop_option_t`

STOP option.

## Enumerator

***SMC\_STOP\_RESERVED*** Reserved stop mode  
***SMC\_STOP1*** Stop with both system and bus clocks disabled  
***SMC\_STOP2*** Stop with system clock disabled and bus clock enabled

Definition at line 142 of file `power_manager_S32K1xx.h`.

## 16.107.4 Function Documentation

16.107.4.1 `status_t POWER_SYS_DoDeinit ( void )`

This function implementation-specific de-initialization of power manager.

This function performs the actual implementation-specific de-initialization.

## Returns

Operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failed.

Definition at line 200 of file `power_manager_S32K1xx.c`.

16.107.4.2 `static void POWER_SYS_DoGetDefaultConfig ( power_manager_user_config_t *const defaultConfig )`  
[inline], [static]

Gets the default power\_manager configuration structure.

This function gets the power\_manager configuration structure of the default power mode.

## Parameters

out	<i>defaultConfig</i>	: Pointer to power mode configuration structure of the default power mode.
-----	----------------------	--

< Power manager mode

< Sleep on exit value

Definition at line 261 of file `power_manager_S32K1xx.h`.

16.107.4.3 `status_t POWER_SYS_DoInit ( void )`

This function implementation-specific configuration of power modes.

This function performs the actual implementation-specific initialization based on the provided power mode configurations. In addition, This function get all clock source were enabled. This one was used for update init clock when CPU jump from very low power mode to run or high speed run mode.

## Returns

Operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failed.

Definition at line 157 of file `power_manager_S32K1xx.c`.

16.107.4.4 `status_t POWER_SYS_DoSetMode ( const power_manager_user_config_t *const configPtr )`

This function configures the power mode.

This function performs the actual implementation-specific logic to switch to one of the defined power modes.

## Parameters

<i>configPtr</i>	Pointer to user configuration structure
------------------	---

## Returns

Operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_MCU\_TRANSITION\_FAILED: Operation failed.

Definition at line 217 of file `power_manager_S32K1xx.c`.

16.107.4.5 `bool POWER_SYS_GetResetSrcStatusCmd ( const RCM_Type *const baseAddr, const rcm_source_names_t srcName )`

Gets the reset source status.

This function gets the current reset source status for a specified source.

#### Parameters

in	<i>baseAddr</i>	Register base address of RCM
in	<i>srcName</i>	reset source name

#### Returns

status True or false for specified reset source

Definition at line 688 of file `power_manager_S32K1xx.c`.



## 16.108 Programmable Delay Block (PDB)

### 16.108.1 Detailed Description

The S32 SDK provides a peripheral driver for the Programmable Delay Block (PDB) module.

The PDB is a configurable counter that can generate events (triggers) that can be used by the ADC to start conversions or routed through TRGMUX to other modules in the device.

#### Modules

- [PDB Driver](#)

*Programmable Delay Block Peripheral Driver.*

## 16.109 Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL)

### 16.109.1 Detailed Description

The S32 SDK provides a Peripheral Abstraction Layer for the PWM mode.

The PWM PAL driver allows to generate PWM signals. It was designed to be portable across all platforms and IPs which support PWM features.

#### How to integrate PWM in your application

Unlike the other drivers, PWM PAL modules need to include a configuration file named `pwm_pal_cfg.h`, which allows the user to specify which IPs are used and how many resources are allocated for each of them (state structures). The following code example shows how to configure one instance for each available PWM IP.

```
#ifndef PWM_PAL_cfg_H
#define PWM_PAL_cfg_H

/* Define which IP instance will be used in current project */
#define PWM_OVER_FTM
#define PWM_OVER_EMIOS
#define PWM_OVER_ETIMER

/* Define the resources necessary for current project */
#define NO_OF_FTM_INSTS_FOR_PWM 1U
#define NO_OF_EMIOS_INSTS_FOR_PWM 1U
#define NO_OF_ETIMER_INSTS_FOR_PWM 1U
#endif /* PWM_PAL_cfg_H */
```

The following table contains the matching between platforms and available IPs

IP/ M↔ CU	S32↔ K116	S32↔ K118	S32↔ K142	S32↔ K144	S32↔ K142↔ W	S32↔ K144↔ W	S32↔ K146	S32↔ K148	M↔ P↔ C5748 G	M↔ P↔ C5746 C	M↔ P↔ C5744 P	S32↔ R274	S32↔ R372
FTM	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	NO	NO	NO	NO	NO
e↔ MI↔ OS	NO	NO	NO	NO	NO	NO	NO	NO	Y↔ ES	Y↔ ES	NO	NO	NO
E↔ TI↔ M↔ ER	NO	NO	NO	NO	NO	NO	NO	NO	NO	NO	Y↔ ES	Y↔ ES	Y↔ ES

In order to use the PWM PAL driver it must be first initialized it using function [PWM\\_Init\(\)](#). Once initialized, it cannot be initialized again for the same PWM module instance until it is de-initialized, using [PWM\\_Deinit\(\)](#). Different PWM module instances can work independently of each other.

After initialization the duty cycle and pwm period can be updated with these functions: [PWM\\_UpdateDuty\(\)](#) and [PWM\\_UpdatePeriod\(\)](#). The measurement unit for duty and period is clock ticks, so the application should be aware about the clock frequency of the timebase used by PWM channel.

Due to hardware limitation period changing for a specific channel can change the period for other channels if they share the same timebase. Also, for FTM all channels must have the same period and type.

#### Important Notes

- The driver enables the interrupts for the corresponding module, but any interrupt priority setting must be done by the application.
- Due to different hardware features is necessary to use different timebase configuration on each platform and some features are available only on some peripherals. To be sure that your applications doesn't try to use unsupported features check return status of called functions and activate `DEV_ERROR_DETECT`.

**Basic code sequence**

1. Initialize PWM\_PAL instance

```
PWM_Init(&pwm_palInstance, &pwm_palConfig);
```

2. Update duty cycle

```
PWM_UpdateDuty(&pwm_palInstance, 0, dutyCycle);
```

3. Update period

```
PWM_UpdatePeriod(&pwm_palInstance, 0, period);
```

4. De-initialize PWM\_PAL instance

```
PWM_Deinit(&pwm_palInstance);
```

**Hardware Limitations****eTimer**

eTimer cannot generate 0% or 100% duty cycles. At least one clock tick will have inverted polarity.

**Integration guideline****Compilation units**

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\pal\src\pwm\pwm_pal.c
```

Additionally, it is required to compile also the .c files from the dependencies listed for each ADC PAL type (please see Dependencies subsection below).

**Include path**

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\pal\inc\  
${S32SDK_PATH}\platform\drivers\inc\
```

An additional file, named *pwm\_pal\_cfg.h*, must be created by the user and added to one of the include paths. The user has to add to the file the definitions of preprocessor symbols according to the PWM PAL type used. These symbols are specified in the next subsection.

When using the S32 SDK configuration tool the file is generated by the configurator.

The pal type PWM\_OVER\_FTM also requires:

```
${S32SDK_PATH}\platform\drivers\src\ftm\
```

**Compile symbols**

Define for selecting one of the PWM PAL type to be used:

```
PWM_OVER_FTM  
PWM_OVER_EMIO  
PWM_OVER_ETIMER
```

## Dependencies

- The pal type PWM\_OVER\_FTM also depends on:  
[FlexTimer Pulse Width Modulation Driver \(FTM\\_PWM\)](#)  
[FlexTimer \(FTM\)](#)
- The pal type PWM\_OVER\_EMIO also depends on:  
[mc\\_emios\\_driver](#)  
[pwm\\_emios\\_driver](#)
- The pal type PWM\_OVER\_ETIMER also depends on:  
[etimer\\_drv](#)

## Data Structures

- struct [pwm\\_ftm\\_timebase\\_t](#)  
*This structure is specific for platforms where FTM is available. Implements : [pwm\\_ftm\\_timebase\\_t\\_Class](#). [More...](#)*
- struct [pwm\\_channel\\_t](#)  
*This structure includes the configuration for each channel Implements : [pwm\\_channel\\_t\\_Class](#). [More...](#)*
- struct [pwm\\_global\\_config\\_t](#)  
*This structure is the configuration for initialization of PWM channels. Implements : [pwm\\_global\\_config\\_t\\_Class](#). [More...](#)*

## Enumerations

- enum [pwm\\_channel\\_type\\_t](#) { PWM\_EDGE\_ALIGNED = 0, PWM\_CENTER\_ALIGNED = 1 }  
*Defines the channel types Implements : [pwm\\_channel\\_type\\_t\\_Class](#).*
- enum [pwm\\_polarity\\_t](#) { PWM\_ACTIVE\_HIGH = 0, PWM\_ACTIVE\_LOW = 1 }  
*Defines the polarity of pwm channels Implements : [pwm\\_polarity\\_t\\_Class](#).*
- enum [pwm\\_complementary\\_mode\\_t](#) { PWM\_DUPLICATED = 0, PWM\_INVERTED = 1 }  
*Defines the polarity of complementary pwm channels relative to main channel Implements : [pwm\\_complementary\\_mode\\_t\\_Class](#).*

## Functions

- status\_t [PWM\\_Init](#) (const [pwm\\_instance\\_t](#) \*const instance, const [pwm\\_global\\_config\\_t](#) \*config)  
*Initialize PWM channels based on config parameter.*
- status\_t [PWM\\_UpdateDuty](#) (const [pwm\\_instance\\_t](#) \*const instance, uint8\_t channel, uint32\_t duty)  
*Update duty cycle. The measurement unit for duty is clock ticks.*
- status\_t [PWM\\_UpdatePeriod](#) (const [pwm\\_instance\\_t](#) \*const instance, uint8\_t channel, uint32\_t period)  
*Update period for specific a specific channel. This function changes period for all channels which shares the timebase with targeted channel.*
- status\_t [PWM\\_OverwriteOutputChannels](#) (const [pwm\\_instance\\_t](#) \*const instance, uint32\_t channelsMask, uint32\_t channelsValues)  
*This function change the output value for some channels. channelsMask select which channels will be overwrite, each bit filed representing one channel: 1 - channel is controlled by channelsValues, 0 - channel is controlled by pwm. channelsValues select output values to be write on corresponding channel. For PWM\_PAL over FTM, when enable complementary channels, if this function is used to force output of complementary channels(n and n+1) with value is high, the output of channel n is going to be high and the output of channel n+1 is going to be low. Please refer to Software ouput control behavior table in the reference manual to get more detail.*

- status\_t [PWM\\_Deinit](#) (const [pwm\\_instance\\_t](#) \*const instance)  
*De-Initialize PWM instance.*

## 16.109.2 Data Structure Documentation

### 16.109.2.1 struct [pwm\\_ftm\\_timebase\\_t](#)

This structure is specific for platforms where FTM is available. Implements : [pwm\\_ftm\\_timebase\\_t\\_Class](#).

Definition at line 92 of file [pwm\\_pal.h](#).

#### Data Fields

- [ftm\\_clock\\_source\\_t](#) sourceClock
- [ftm\\_clock\\_ps\\_t](#) prescaler
- [ftm\\_deadtime\\_ps\\_t](#) deadtimePrescaler

#### Field Documentation

##### 16.109.2.1.1 [ftm\\_deadtime\\_ps\\_t](#) deadtimePrescaler

Prescaler for FTM dead-time insertion

Definition at line 96 of file [pwm\\_pal.h](#).

##### 16.109.2.1.2 [ftm\\_clock\\_ps\\_t](#) prescaler

Prescaler for FTM timebase

Definition at line 95 of file [pwm\\_pal.h](#).

##### 16.109.2.1.3 [ftm\\_clock\\_source\\_t](#) sourceClock

Clock source for FTM timebase

Definition at line 94 of file [pwm\\_pal.h](#).

### 16.109.2.2 struct [pwm\\_channel\\_t](#)

This structure includes the configuration for each channel Implements : [pwm\\_channel\\_t\\_Class](#).

Definition at line 147 of file [pwm\\_pal.h](#).

#### Data Fields

- [uint8\\_t](#) channel
- [pwm\\_channel\\_type\\_t](#) channelType
- [uint32\\_t](#) period
- [uint32\\_t](#) duty
- [pwm\\_polarity\\_t](#) polarity
- bool insertDeadtime
- [uint8\\_t](#) deadtime
- bool enableComplementaryChannel
- [pwm\\_complementary\\_mode\\_t](#) complementaryChannelPolarity
- void \* timebase

#### Field Documentation

##### 16.109.2.2.1 [uint8\\_t](#) channel

Channel number

Definition at line 149 of file [pwm\\_pal.h](#).

**16.109.2.2.2 pwm\_channel\_type\_t channelType**

Channel waveform type

Definition at line 150 of file pwm\_pal.h.

**16.109.2.2.3 pwm\_complementary\_mode\_t complementaryChannelPolarity**

Configure the polarity of the complementary channel relative to the main channel

Definition at line 157 of file pwm\_pal.h.

**16.109.2.2.4 uint8\_t deadtime**

Dead-time value in ticks

Definition at line 155 of file pwm\_pal.h.

**16.109.2.2.5 uint32\_t duty**

Duty cycle in ticks

Definition at line 152 of file pwm\_pal.h.

**16.109.2.2.6 bool enableComplementaryChannel**

Enable a complementary channel. This option can take control over other channel than the channel configured in this structure.

Definition at line 156 of file pwm\_pal.h.

**16.109.2.2.7 bool insertDeadtime**

Enable/disable dead-time insertion. This feature is available only if complementary mode is enabled

Definition at line 154 of file pwm\_pal.h.

**16.109.2.2.8 uint32\_t period**

Period of the PWM signal in ticks

Definition at line 151 of file pwm\_pal.h.

**16.109.2.2.9 pwm\_polarity\_t polarity**

Channel polarity

Definition at line 153 of file pwm\_pal.h.

**16.109.2.2.10 void\* timebase**

This field is platform specific and it's used to configure the clocking tree for different time-bases. If FTM is use this field must be filled by a pointer to [pwm\\_ftm\\_timebase\\_t](#)

Definition at line 158 of file pwm\_pal.h.

**16.109.2.3 struct pwm\_global\_config\_t**

This structure is the configuration for initialization of PWM channels. Implements : [pwm\\_global\\_config\\_t\\_Class](#).

Definition at line 166 of file pwm\_pal.h.

**Data Fields**

- [pwm\\_channel\\_t](#) \* [pwmChannels](#)
- [uint8\\_t](#) [numberOfPwmChannels](#)

## Field Documentation

16.109.2.3.1 `uint8_t numberOfPwmChannels`

Number of channels which are configured

Definition at line 169 of file `pwm_pal.h`.

16.109.2.3.2 `pwm_channel_t* pwmChannels`

Pointer to channels configurations

Definition at line 168 of file `pwm_pal.h`.

## 16.109.3 Enumeration Type Documentation

16.109.3.1 `enum pwm_channel_type_t`

Defines the channel types Implements : `pwm_channel_type_t_Class`.

## Enumerator

***PWM\_EDGE\_ALIGNED*** Counter used by this type of channel is in up counting mode and the edge is aligned to PWM period

***PWM\_CENTER\_ALIGNED*** Counter used by this type of channel is in up-down counting mode and the duty is inserted in center of PWM period

Definition at line 60 of file `pwm_pal.h`.

16.109.3.2 `enum pwm_complementary_mode_t`

Defines the polarity of complementary pwm channels relative to main channel Implements : `pwm_complementary_mode_t_Class`.

## Enumerator

***PWM\_DUPLICATED*** Complementary channel is the same as main channel

***PWM\_INVERTED*** Complementary channel is inverted relative to main channel

Definition at line 80 of file `pwm_pal.h`.

16.109.3.3 `enum pwm_polarity_t`

Defines the polarity of pwm channels Implements : `pwm_polarity_t_Class`.

## Enumerator

***PWM\_ACTIVE\_HIGH*** Polarity is active high

***PWM\_ACTIVE\_LOW*** Polarity is active low

Definition at line 70 of file `pwm_pal.h`.

## 16.109.4 Function Documentation

16.109.4.1 `status_t PWM_Deinit ( const pwm_instance_t *const instance )`

De-Initialize PWM instance.

**Parameters**

<i>in</i>	<i>instance</i>	The name of the instance
-----------	-----------------	--------------------------

**Returns**

Error or success status returned by API

Definition at line 819 of file pwm\_pal.c.

**16.109.4.2** `status_t PWM_Init ( const pwm_instance_t *const instance, const pwm_global_config_t * config )`

Initialize PWM channels based on config parameter.

**Parameters**

<i>in</i>	<i>instance</i>	The name of the instance
<i>in</i>	<i>config</i>	The configuration structure used to initialize PWM modules

**Returns**

Error or success status returned by API

Definition at line 133 of file pwm\_pal.c.

**16.109.4.3** `status_t PWM_OverwriteOutputChannels ( const pwm_instance_t *const instance, uint32_t channelsMask, uint32_t channelsValues )`

This function change the output value for some channels. channelsMask select which channels will be overwrite, each bit filed representing one channel: 1 - channel is controlled by channelsValues, 0 - channel is controlled by pwm. channelsValues select output values to be write on corresponding channel. For PWM\_PAL over FTM, when enable complementary channels, if this function is used to force output of complementary channels(n and n+1) with value is high, the output of channel n is going to be high and the output of channel n+1 is going to be low. Please refer to Software ouput control behavior table in the reference manual to get more detail.

**Parameters**

<i>in</i>	<i>instance</i>	The name of the instance
<i>in</i>	<i>channelsMask</i>	The name mask used to select which channel is overwrite
<i>in</i>	<i>channelsValues</i>	The name overwrite values for all channels

**Returns**

Error or success status returned by API

Definition at line 775 of file pwm\_pal.c.

**16.109.4.4** `status_t PWM_UpdateDuty ( const pwm_instance_t *const instance, uint8_t channel, uint32_t duty )`

Update duty cycle. The measurement unit for duty is clock ticks.

**Parameters**

<i>in</i>	<i>instance</i>	The name of the instance
<i>in</i>	<i>channel</i>	The channel which is update
<i>in</i>	<i>duty</i>	The duty cycle measured in ticks

**Returns**

Error or success status returned by API

Definition at line 574 of file pwm\_pal.c.



16.109.4.5 `status_t PWM_UpdatePeriod ( const pwm_instance_t *const instance, uint8_t channel, uint32_t period )`

Update period for specific a specific channel. This function changes period for all channels which shares the timebase with targeted channel.

**Parameters**

<i>in</i>	<i>instance</i>	The name of the instance
<i>in</i>	<i>channel</i>	The channel which is update
<i>in</i>	<i>period</i>	The period measured in ticks

**Returns**

Error or success status returned by API

Definition at line 669 of file pwm\_pal.c.

## 16.110 RTC Driver

### 16.110.1 Detailed Description

Real Time Clock Peripheral Driver.

This section describes the programming interface of the RTC driver.

#### Data Structures

- struct [rtc\\_timedate\\_t](#)  
RTC Time Date structure Implements : [rtc\\_timedate\\_t\\_Class](#). [More...](#)
- struct [rtc\\_init\\_config\\_t](#)  
RTC Initialization structure Implements : [rtc\\_init\\_config\\_t\\_Class](#). [More...](#)
- struct [rtc\\_alarm\\_config\\_t](#)  
RTC alarm configuration Implements : [rtc\\_alarm\\_config\\_t\\_Class](#). [More...](#)
- struct [rtc\\_interrupt\\_config\\_t](#)  
RTC interrupt configuration. It is used to configure interrupt other than Time Alarm and Time Seconds interrupt Implements : [rtc\\_interrupt\\_config\\_t\\_Class](#). [More...](#)
- struct [rtc\\_seconds\\_int\\_config\\_t](#)  
RTC Seconds Interrupt Configuration Implements : [rtc\\_seconds\\_int\\_config\\_t\\_Class](#). [More...](#)
- struct [rtc\\_register\\_lock\\_config\\_t](#)  
RTC Register Lock Configuration Implements : [rtc\\_register\\_lock\\_config\\_t\\_Class](#). [More...](#)

#### Macros

- #define [SECONDS\\_IN\\_A\\_DAY](#) (86400UL)
- #define [SECONDS\\_IN\\_A\\_HOUR](#) (3600U)
- #define [SECONDS\\_IN\\_A\\_MIN](#) (60U)
- #define [MINS\\_IN\\_A\\_HOUR](#) (60U)
- #define [HOURS\\_IN\\_A\\_DAY](#) (24U)
- #define [DAYS\\_IN\\_A\\_YEAR](#) (365U)
- #define [DAYS\\_IN\\_A\\_LEAP\\_YEAR](#) (366U)
- #define [YEAR\\_RANGE\\_START](#) (1970U)
- #define [YEAR\\_RANGE\\_END](#) (2099U)

#### Enumerations

- enum [rtc\\_second\\_int\\_cfg\\_t](#) {  
[RTC\\_INT\\_1HZ](#) = 0x00U, [RTC\\_INT\\_2HZ](#) = 0x01U, [RTC\\_INT\\_4HZ](#) = 0x02U, [RTC\\_INT\\_8HZ](#) = 0x03U,  
[RTC\\_INT\\_16HZ](#) = 0x04U, [RTC\\_INT\\_32HZ](#) = 0x05U, [RTC\\_INT\\_64HZ](#) = 0x06U, [RTC\\_INT\\_128HZ](#) = 0x07U }  
RTC Seconds interrupt configuration Implements : [rtc\\_second\\_int\\_cfg\\_t\\_Class](#).
- enum [rtc\\_clk\\_out\\_config\\_t](#) { [RTC\\_CLKOUT\\_DISABLED](#) = 0x00U, [RTC\\_CLKOUT\\_SRC\\_TSIC](#) = 0x01U, [RTC\\_CLKOUT\\_SRC\\_32KHZ](#) = 0x02U }  
RTC CLKOUT pin configuration Implements : [rtc\\_clk\\_out\\_config\\_t\\_Class](#).
- enum [rtc\\_clk\\_select\\_t](#) { [RTC\\_CLK\\_SRC\\_OSC\\_32KHZ](#) = 0x00U, [RTC\\_CLK\\_SRC\\_LPO\\_1KHZ](#) = 0x01U }  
RTC clock select Implements : [rtc\\_clk\\_select\\_t\\_Class](#).
- enum [rtc\\_lock\\_register\\_select\\_t](#) { [RTC\\_LOCK\\_REG\\_LOCK](#) = 0x00U, [RTC\\_STATUS\\_REG\\_LOCK](#) = 0x01U,  
[RTC\\_CTRL\\_REG\\_LOCK](#) = 0x02U, [RTC\\_TCL\\_REG\\_LOCK](#) = 0x03U }  
RTC register lock Implements : [rtc\\_lock\\_register\\_select\\_t\\_Class](#).

## Functions

- status\_t [RTC\\_DRV\\_Init](#) (uint32\_t instance, const [rtc\\_init\\_config\\_t](#) \*const rtcUserCfg)
 

*This function initializes the RTC instance with the settings provided by the user via the [rtcUserCfg](#) parameter. The user must ensure that clock is enabled for the RTC instance used. If the Control register is locked then this method returns [STATUS\\_ERROR](#). In order to clear the CR Lock the user must perform a power-on reset.*
- status\_t [RTC\\_DRV\\_Deinit](#) (uint32\_t instance)
 

*This function deinitializes the RTC instance. If the Control register is locked then this method returns [STATUS\\_ERROR](#).*
- void [RTC\\_DRV\\_GetDefaultConfig](#) ([rtc\\_init\\_config\\_t](#) \*const config)
 

*This function will set the default configuration values into the structure passed as a parameter.*
- status\_t [RTC\\_DRV\\_StartCounter](#) (uint32\_t instance)
 

*Start RTC instance counter. Before calling this function the user should use [RTC\\_DRV\\_SetTimeDate](#) to configure the start time.*
- status\_t [RTC\\_DRV\\_StopCounter](#) (uint32\_t instance)
 

*Disable RTC instance counter.*
- status\_t [RTC\\_DRV\\_GetCurrentTimeDate](#) (uint32\_t instance, [rtc\\_timedate\\_t](#) \*const currentTime)
 

*Get current time and date from RTC instance.*
- status\_t [RTC\\_DRV\\_SetTimeDate](#) (uint32\_t instance, const [rtc\\_timedate\\_t](#) \*const time)
 

*Set time and date for RTC instance. The user must stop the counter before using this function. Otherwise it will return an error.*
- status\_t [RTC\\_DRV\\_ConfigureRegisterLock](#) (uint32\_t instance, const [rtc\\_register\\_lock\\_config\\_t](#) \*const lockConfig)
 

*This method configures register lock for the corresponding RTC instance. Remember that all the registers are unlocked only by software reset or power on reset. (Except for CR that is unlocked only by POR).*
- void [RTC\\_DRV\\_GetRegisterLock](#) (uint32\_t instance, [rtc\\_register\\_lock\\_config\\_t](#) \*const lockConfig)
 

*Get which registers are locked for RTC instance.*
- status\_t [RTC\\_DRV\\_ConfigureTimeCompensation](#) (uint32\_t instance, uint8\_t complInterval, int8\_t compensation)
 

*This method configures time compensation. Data is passed by the [complInterval](#) and [compensation](#) parameters. For more details regarding coefficient calculation see the Reference Manual.*
- void [RTC\\_DRV\\_GetTimeCompensation](#) (uint32\_t instance, uint8\_t \*complInterval, int8\_t \*compensation)
 

*This retrieves the time compensation coefficients and saves them on the variables referenced by the parameters.*
- void [RTC\\_DRV\\_ConfigureFaultInt](#) (uint32\_t instance, [rtc\\_interrupt\\_config\\_t](#) \*const intConfig)
 

*This method configures fault interrupts such as:*
- void [RTC\\_DRV\\_ConfigureSecondsInt](#) (uint32\_t instance, [rtc\\_seconds\\_int\\_config\\_t](#) \*const intConfig)
 

*This method configures the Time Seconds Interrupt with the configuration from the [intConfig](#) parameter.*
- status\_t [RTC\\_DRV\\_ConfigureAlarm](#) (uint32\_t instance, [rtc\\_alarm\\_config\\_t](#) \*const alarmConfig)
 

*This method configures the alarm with the configuration from the [alarmConfig](#) parameter.*
- void [RTC\\_DRV\\_GetAlarmConfig](#) (uint32\_t instance, [rtc\\_alarm\\_config\\_t](#) \*alarmConfig)
 

*Get alarm configuration for RTC instance.*
- bool [RTC\\_DRV\\_IsAlarmPending](#) (uint32\_t instance)
 

*Check if alarm is pending.*
- void [RTC\\_DRV\\_ConvertSecondsToTimeDate](#) (const uint32\_t \*seconds, [rtc\\_timedate\\_t](#) \*const timeDate)
 

*Convert seconds to [rtc\\_timedate\\_t](#) structure.*
- void [RTC\\_DRV\\_ConvertTimeDateToSeconds](#) (const [rtc\\_timedate\\_t](#) \*const timeDate, uint32\_t \*const seconds)
 

*Convert seconds to [rtc\\_timedate\\_t](#) structure.*
- bool [RTC\\_DRV\\_IsYearLeap](#) (uint16\_t year)
 

*Check if the current year is leap.*
- bool [RTC\\_DRV\\_IsTimeDateCorrectFormat](#) (const [rtc\\_timedate\\_t](#) \*const timeDate)
 

*Check if the date time struct is configured properly.*
- status\_t [RTC\\_DRV\\_GetNextAlarmTime](#) (uint32\_t instance, [rtc\\_timedate\\_t](#) \*const alarmTime)

*Gets the next alarm time.*

- void [RTC\\_DRV\\_IRQHandler](#) (uint32\_t instance)

*This method is the API's Interrupt handler for generic and alarm IRQ. It will handle the alarm repetition and calls the user callbacks if they are not NULL.*

- void [RTC\\_DRV\\_SecondsIRQHandler](#) (uint32\_t instance)

*This method is the API's Interrupt handler for RTC Second interrupt. This ISR will call the user callback if defined.*

## 16.110.2 Data Structure Documentation

### 16.110.2.1 struct rtc\_timedate\_t

RTC Time Date structure Implements : [rtc\\_timedate\\_t\\_Class](#).

Definition at line 97 of file [rtc\\_driver.h](#).

#### Data Fields

- uint16\_t [year](#)
- uint16\_t [month](#)
- uint16\_t [day](#)
- uint16\_t [hour](#)
- uint16\_t [minutes](#)
- uint8\_t [seconds](#)

#### Field Documentation

##### 16.110.2.1.1 uint16\_t day

Day

Definition at line 101 of file [rtc\\_driver.h](#).

##### 16.110.2.1.2 uint16\_t hour

Hour

Definition at line 102 of file [rtc\\_driver.h](#).

##### 16.110.2.1.3 uint16\_t minutes

Minutes

Definition at line 103 of file [rtc\\_driver.h](#).

##### 16.110.2.1.4 uint16\_t month

Month

Definition at line 100 of file [rtc\\_driver.h](#).

##### 16.110.2.1.5 uint8\_t seconds

Seconds

Definition at line 104 of file [rtc\\_driver.h](#).

##### 16.110.2.1.6 uint16\_t year

Year

Definition at line 99 of file [rtc\\_driver.h](#).

### 16.110.2.2 struct rtc\_init\_config\_t

RTC Initialization structure Implements : rtc\_init\_config\_t\_Class.

Definition at line 111 of file rtc\_driver.h.

#### Data Fields

- uint8\_t [compensationInterval](#)
- int8\_t [compensation](#)
- [rtc\\_clk\\_select\\_t](#) clockSelect
- [rtc\\_clk\\_out\\_config\\_t](#) clockOutConfig
- bool [updateEnable](#)
- bool [nonSupervisorAccessEnable](#)

#### Field Documentation

#### 16.110.2.2.1 rtc\_clk\_out\_config\_t clockOutConfig

RTC Clock Out Source

Definition at line 116 of file rtc\_driver.h.

#### 16.110.2.2.2 rtc\_clk\_select\_t clockSelect

RTC Clock Select

Definition at line 115 of file rtc\_driver.h.

#### 16.110.2.2.3 int8\_t compensation

Compensation Value

Definition at line 114 of file rtc\_driver.h.

#### 16.110.2.2.4 uint8\_t compensationInterval

Compensation Interval

Definition at line 113 of file rtc\_driver.h.

#### 16.110.2.2.5 bool nonSupervisorAccessEnable

Enable writes to the registers in non Supervisor Mode

Definition at line 118 of file rtc\_driver.h.

#### 16.110.2.2.6 bool updateEnable

Enable changing the Time Counter Enable bit even if the Status register is locked

Definition at line 117 of file rtc\_driver.h.

### 16.110.2.3 struct rtc\_alarm\_config\_t

RTC alarm configuration Implements : rtc\_alarm\_config\_t\_Class.

Definition at line 125 of file rtc\_driver.h.

#### Data Fields

- [rtc\\_timedate\\_t](#) alarmTime
- uint32\_t [repetitionInterval](#)
- uint32\_t [numberOfRepeats](#)
- bool [repeatForever](#)

- bool [alarmIntEnable](#)
- void(\* [alarmCallback](#) )(void \*callbackParam)
- void \* [callbackParams](#)

#### Field Documentation

##### 16.110.2.3.1 void(\* alarmCallback) (void \*callbackParam)

Pointer to the user callback method.

Definition at line 132 of file rtc\_driver.h.

##### 16.110.2.3.2 bool alarmIntEnable

Enable alarm interrupt

Definition at line 131 of file rtc\_driver.h.

##### 16.110.2.3.3 rtc\_timedate\_t alarmTime

Alarm time

Definition at line 127 of file rtc\_driver.h.

##### 16.110.2.3.4 void\* callbackParams

Pointer to the callback parameters.

Definition at line 133 of file rtc\_driver.h.

##### 16.110.2.3.5 uint32\_t numberOfRepeats

Number of alarm repeats

Definition at line 129 of file rtc\_driver.h.

##### 16.110.2.3.6 bool repeatForever

Repeat forever if set, discard number of repeats

Definition at line 130 of file rtc\_driver.h.

##### 16.110.2.3.7 uint32\_t repetitionInterval

Interval of repetition in sec

Definition at line 128 of file rtc\_driver.h.

##### 16.110.2.4 struct rtc\_interrupt\_config\_t

RTC interrupt configuration. It is used to configure interrupt other than Time Alarm and Time Seconds interrupt  
Implements : rtc\_interrupt\_config\_t\_Class.

Definition at line 141 of file rtc\_driver.h.

#### Data Fields

- bool [overflowIntEnable](#)
- bool [timeInvalidIntEnable](#)
- void(\* [rtcCallback](#) )(void \*callbackParam)
- void \* [callbackParams](#)

#### Field Documentation

**16.110.2.4.1 void\* callbackParams**

Pointer to the callback parameters.

Definition at line 146 of file rtc\_driver.h.

**16.110.2.4.2 bool overflowIntEnable**

Enable Time Overflow Interrupt

Definition at line 143 of file rtc\_driver.h.

**16.110.2.4.3 void(\* rtcCallback)(void \*callbackParam)**

Pointer to the user callback method.

Definition at line 145 of file rtc\_driver.h.

**16.110.2.4.4 bool timeInvalidIntEnable**

Enable Time Invalid Interrupt

Definition at line 144 of file rtc\_driver.h.

**16.110.2.5 struct rtc\_seconds\_int\_config\_t**

RTC Seconds Interrupt Configuration Implements : rtc\_seconds\_int\_config\_t\_Class.

Definition at line 153 of file rtc\_driver.h.

**Data Fields**

- [rtc\\_second\\_int\\_cfg\\_t secondIntConfig](#)
- bool [secondIntEnable](#)
- void(\* [rtcSecondsCallback](#))(void \*callbackParam)
- void \* [secondsCallbackParams](#)

**Field Documentation****16.110.2.5.1 void(\* rtcSecondsCallback)(void \*callbackParam)**

Pointer to the user callback method.

Definition at line 157 of file rtc\_driver.h.

**16.110.2.5.2 rtc\_second\_int\_cfg\_t secondIntConfig**

Seconds Interrupt frequency

Definition at line 155 of file rtc\_driver.h.

**16.110.2.5.3 bool secondIntEnable**

Seconds Interrupt enable

Definition at line 156 of file rtc\_driver.h.

**16.110.2.5.4 void\* secondsCallbackParams**

Pointer to the callback parameters.

Definition at line 158 of file rtc\_driver.h.

**16.110.2.6 struct rtc\_register\_lock\_config\_t**

RTC Register Lock Configuration Implements : rtc\_register\_lock\_config\_t\_Class.



Definition at line 165 of file rtc\_driver.h.

#### Data Fields

- bool [lockRegisterLock](#)
- bool [statusRegisterLock](#)
- bool [controlRegisterLock](#)
- bool [timeCompensationRegisterLock](#)

#### Field Documentation

##### 16.110.2.6.1 bool controlRegisterLock

Lock state of the Control Register

Definition at line 169 of file rtc\_driver.h.

##### 16.110.2.6.2 bool lockRegisterLock

Lock state of the Lock Register

Definition at line 167 of file rtc\_driver.h.

##### 16.110.2.6.3 bool statusRegisterLock

Lock state of the Status Register

Definition at line 168 of file rtc\_driver.h.

##### 16.110.2.6.4 bool timeCompensationRegisterLock

Lock state of the Time Compensation Register

Definition at line 170 of file rtc\_driver.h.

#### 16.110.3 Macro Definition Documentation

##### 16.110.3.1 #define DAYS\_IN\_A\_LEAP\_YEAR (366U)

Definition at line 40 of file rtc\_driver.h.

##### 16.110.3.2 #define DAYS\_IN\_A\_YEAR (365U)

Definition at line 39 of file rtc\_driver.h.

##### 16.110.3.3 #define HOURS\_IN\_A\_DAY (24U)

Definition at line 38 of file rtc\_driver.h.

##### 16.110.3.4 #define MINS\_IN\_A\_HOUR (60U)

Definition at line 37 of file rtc\_driver.h.

##### 16.110.3.5 #define SECONDS\_IN\_A\_DAY (86400UL)

Definition at line 34 of file rtc\_driver.h.

##### 16.110.3.6 #define SECONDS\_IN\_A\_HOUR (3600U)

Definition at line 35 of file rtc\_driver.h.

#### 16.110.3.7 #define SECONDS\_IN\_A\_MIN (60U)

Definition at line 36 of file rtc\_driver.h.

#### 16.110.3.8 #define YEAR\_RANGE\_END (2099U)

Definition at line 42 of file rtc\_driver.h.

#### 16.110.3.9 #define YEAR\_RANGE\_START (1970U)

Definition at line 41 of file rtc\_driver.h.

### 16.110.4 Enumeration Type Documentation

#### 16.110.4.1 enum rtc\_clk\_out\_config\_t

RTC CLKOUT pin configuration Implements : rtc\_clk\_out\_config\_t\_Class.

##### Enumerator

- RTC\_CLKOUT\_DISABLED** Clock out pin is disabled
- RTC\_CLKOUT\_SRC\_TSIC** Output on RTC\_CLKOUT as configured on Time seconds interrupt
- RTC\_CLKOUT\_SRC\_32KHZ** Output on RTC\_CLKOUT of the 32KHz clock

Definition at line 64 of file rtc\_driver.h.

#### 16.110.4.2 enum rtc\_clk\_select\_t

RTC clock select Implements : rtc\_clk\_select\_t\_Class.

##### Enumerator

- RTC\_CLK\_SRC\_OSC\_32KHZ** RTC Prescaler increments using 32 KHz crystal
- RTC\_CLK\_SRC\_LPO\_1KHZ** RTC Prescaler increments using 1KHz LPO

Definition at line 75 of file rtc\_driver.h.

#### 16.110.4.3 enum rtc\_lock\_register\_select\_t

RTC register lock Implements : rtc\_lock\_register\_select\_t\_Class.

##### Enumerator

- RTC\_LOCK\_REG\_LOCK** RTC Lock Register lock
- RTC\_STATUS\_REG\_LOCK** RTC Status Register lock
- RTC\_CTRL\_REG\_LOCK** RTC Control Register lock
- RTC\_TCL\_REG\_LOCK** RTC Time Compensation Reg lock

Definition at line 85 of file rtc\_driver.h.

#### 16.110.4.4 enum rtc\_second\_int\_cfg\_t

RTC Seconds interrupt configuration Implements : rtc\_second\_int\_cfg\_t\_Class.

##### Enumerator

- RTC\_INT\_1HZ** RTC seconds interrupt occurs at 1 Hz
- RTC\_INT\_2HZ** RTC seconds interrupt occurs at 2 Hz

**RTC\_INT\_4HZ** RTC seconds interrupt occurs at 4 Hz

**RTC\_INT\_8HZ** RTC seconds interrupt occurs at 8 Hz

**RTC\_INT\_16HZ** RTC seconds interrupt occurs at 16 Hz

**RTC\_INT\_32HZ** RTC seconds interrupt occurs at 32 Hz

**RTC\_INT\_64HZ** RTC seconds interrupt occurs at 64 Hz

**RTC\_INT\_128HZ** RTC seconds interrupt occurs at 128 Hz

Definition at line 48 of file rtc\_driver.h.

#### 16.110.5 Function Documentation

**16.110.5.1** `status_t RTC_DRV_ConfigureAlarm ( uint32_t instance, rtc_alarm_config_t *const alarmConfig )`

This method configures the alarm with the configuration from the alarmConfig parameter.

##### Parameters

in	<i>instance</i>	The number of the RTC instance used
in	<i>alarmConfig</i>	Pointer to the structure which holds the alarm configuration

##### Returns

STATUS\_SUCCESS if the configuration is successful or STATUS\_ERROR if the alarm time is invalid.

Definition at line 933 of file rtc\_driver.c.

**16.110.5.2** `void RTC_DRV_ConfigureFaultInt ( uint32_t instance, rtc_interrupt_config_t *const intConfig )`

This method configures fault interrupts such as:

- Time Overflow Interrupt
- Time Invalid Interrupt with the user provided configuration struct intConfig.

##### Parameters

in	<i>instance</i>	The number of the RTC instance used
in	<i>intConfig</i>	Pointer to the structure which holds the configuration

##### Returns

None

Definition at line 876 of file rtc\_driver.c.

**16.110.5.3** `status_t RTC_DRV_ConfigureRegisterLock ( uint32_t instance, const rtc_register_lock_config_t *const lockConfig )`

This method configures register lock for the corresponding RTC instance. Remember that all the registers are unlocked only by software reset or power on reset. (Except for CR that is unlocked only by POR).

##### Parameters

in	<i>instance</i>	The number of the RTC instance used
in	<i>lockConfig</i>	Pointer to the lock configuration structure

**Returns**

STATUS\_SUCCESS if the operation was successful, STATUS\_ERROR if the Lock Register is locked.

Definition at line 426 of file rtc\_driver.c.

**16.110.5.4** void RTC\_DRV\_ConfigureSecondsInt ( uint32\_t *instance*, rtc\_seconds\_int\_config\_t \*const *intConfig* )

This method configures the Time Seconds Interrupt with the configuration from the intConfig parameter.

**Parameters**

in	<i>instance</i>	The number of the RTC instance used
in	<i>intConfig</i>	Pointer to the structure which holds the configuration

**Returns**

None

Definition at line 903 of file rtc\_driver.c.

**16.110.5.5** status\_t RTC\_DRV\_ConfigureTimeCompensation ( uint32\_t *instance*, uint8\_t *complInterval*, int8\_t *compensation* )

This method configures time compensation. Data is passed by the complInterval and compensation parameters. For more details regarding coefficient calculation see the Reference Manual.

**Parameters**

in	<i>instance</i>	The number of the RTC instance used
in	<i>complInterval</i>	Compensation interval
in	<i>compensation</i>	Compensation value

**Returns**

STATUS\_SUCCESS if the operation was successful, STATUS\_ERROR if the TC Register is locked.

Definition at line 501 of file rtc\_driver.c.

**16.110.5.6** void RTC\_DRV\_ConvertSecondsToTimeDate ( const uint32\_t \* *seconds*, rtc\_timedate\_t \*const *timeDate* )

Convert seconds to [rtc\\_timedate\\_t](#) structure.

**Parameters**

in	<i>seconds</i>	Pointer to the seconds
out	<i>timeDate</i>	Pointer to the structure in which to store the result

**Returns**

None

Definition at line 551 of file rtc\_driver.c.

**16.110.5.7** void RTC\_DRV\_ConvertTimeDateToSeconds ( const rtc\_timedate\_t \*const *timeDate*, uint32\_t \*const *seconds* )

Convert seconds to [rtc\\_timedate\\_t](#) structure.

**Parameters**

in	<i>timeDate</i>	Pointer to the source struct
out	<i>seconds</i>	Pointer to the variable in which to store the result

**Returns**

None

Definition at line 645 of file rtc\_driver.c.

**16.110.5.8 status\_t RTC\_DRV\_Deinit ( uint32\_t instance )**

This function deinitializes the RTC instance. If the Control register is locked then this method returns STATUS\_ERROR.

**Parameters**

in	<i>instance</i>	The number of the RTC instance used
----	-----------------	-------------------------------------

**Returns**

STATUS\_SUCCESS if the operation was successful or STATUS\_ERROR if Control register is locked.

Definition at line 158 of file rtc\_driver.c.

**16.110.5.9 void RTC\_DRV\_GetAlarmConfig ( uint32\_t instance, rtc\_alarm\_config\_t \* alarmConfig )**

Get alarm configuration for RTC instance.

**Parameters**

in	<i>instance</i>	The number of the RTC instance used
out	<i>alarmConfig</i>	Pointer to the structure in which to store the alarm configuration

**Returns**

None

Definition at line 987 of file rtc\_driver.c.

**16.110.5.10 status\_t RTC\_DRV\_GetCurrentTimeDate ( uint32\_t instance, rtc\_timedate\_t \*const currentTime )**

Get current time and date from RTC instance.

**Parameters**

in	<i>instance</i>	The number of the RTC instance used
out	<i>currentTime</i>	Pointer to the variable in which to store the result

**Returns**

STATUS\_SUCCESS if the operation was successful, STATUS\_ERROR if there was a problem.

Definition at line 327 of file rtc\_driver.c.

**16.110.5.11 void RTC\_DRV\_GetDefaultConfig ( rtc\_init\_config\_t \*const config )**

This function will set the default configuration values into the structure passed as a parameter.

**Parameters**

out	<i>config</i>	Pointer to the structure in which the configuration will be saved.
-----	---------------	--

**Returns**

None

Definition at line 194 of file rtc\_driver.c.

**16.110.5.12** `status_t RTC_DRV_GetNextAlarmTime ( uint32_t instance, rtc_timedate_t *const alarmTime )`

Gets the next alarm time.

**Parameters**

in	<i>instance</i>	The number of the RTC instance used
out	<i>alarmTime</i>	Pointer to the variable in which to store the data

**Returns**

STATUS\_SUCCESS if the next alarm time is valid, STATUS\_ERROR if there is no new alarm or alarm configuration specified.

Definition at line 1019 of file rtc\_driver.c.

**16.110.5.13** `void RTC_DRV_GetRegisterLock ( uint32_t instance, rtc_register_lock_config_t *const lockConfig )`

Get which registers are locked for RTC instance.

**Parameters**

in	<i>instance</i>	The number of the RTC instance used
out	<i>lockConfig</i>	Pointer to the lock configuration structure in which to save the data

**Returns**

None

Definition at line 473 of file rtc\_driver.c.

**16.110.5.14** `void RTC_DRV_GetTimeCompensation ( uint32_t instance, uint8_t * complInterval, int8_t * compensation )`

This retrieves the time compensation coefficients and saves them on the variables referenced by the parameters.

**Parameters**

in	<i>instance</i>	The number of the RTC instance used
out	<i>complInterval</i>	Pointer to the variable in which to save the compensation interval
out	<i>compensation</i>	Pointer to the variable in which to save the compensation value

**Returns**

None

Definition at line 534 of file rtc\_driver.c.

**16.110.5.15** `status_t RTC_DRV_Init ( uint32_t instance, const rtc_init_config_t *const rtcUserCfg )`

This function initializes the RTC instance with the settings provided by the user via the rtcUserCfg parameter. The user must ensure that clock is enabled for the RTC instance used. If the Control register is locked then this method returns STATUS\_ERROR. In order to clear the CR Lock the user must perform a power-on reset.

**Parameters**

in	<i>instance</i>	The number of the RTC instance used
in	<i>rtcUserCfg</i>	Pointer to the user's configuration structure

**Returns**

STATUS\_SUCCESS if the operation was successful, STATUS\_ERROR if Control is locked.

Definition at line 97 of file rtc\_driver.c.

**16.110.5.16 void RTC\_DRV\_IRQHandler ( uint32\_t *instance* )**

This method is the API's Interrupt handler for generic and alarm IRQ. It will handle the alarm repetition and calls the user callbacks if they are not NULL.

**Parameters**

in	<i>instance</i>	RTC instance used
----	-----------------	-------------------

**Returns**

None

Definition at line 773 of file rtc\_driver.c.

**16.110.5.17 bool RTC\_DRV\_IsAlarmPending ( uint32\_t *instance* )**

Check if alarm is pending.

**Parameters**

in	<i>instance</i>	The number of the RTC instance used
----	-----------------	-------------------------------------

**Returns**

True if the alarm has occurred, false if not

Definition at line 1002 of file rtc\_driver.c.

**16.110.5.18 bool RTC\_DRV\_IsTimeDateCorrectFormat ( const rtc\_timedate\_t \*const *timeDate* )**

Check if the date time struct is configured properly.

**Parameters**

in	<i>timeDate</i>	Structure to check to check
----	-----------------	-----------------------------

**Returns**

True if the time date is in the correct format, false if not

Definition at line 695 of file rtc\_driver.c.

**16.110.5.19 bool RTC\_DRV\_IsYearLeap ( uint16\_t *year* )**

Check if the current year is leap.

**Parameters**

<i>in</i>	<i>year</i>	Year to check
-----------	-------------	---------------

**Returns**

True if the year is leap, false if not

Definition at line 737 of file rtc\_driver.c.

**16.110.5.20** void RTC\_DRV\_SecondsIRQHandler ( uint32\_t *instance* )

This method is the API's Interrupt handler for RTC Second interrupt. This ISR will call the user callback if defined.

**Parameters**

<i>in</i>	<i>instance</i>	RTC instance used
-----------	-----------------	-------------------

**Returns**

None

Definition at line 850 of file rtc\_driver.c.

**16.110.5.21** status\_t RTC\_DRV\_SetTimeDate ( uint32\_t *instance*, const rtc\_timedate\_t \*const *time* )

Set time and date for RTC instance. The user must stop the counter before using this function. Otherwise it will return an error.

**Parameters**

<i>in</i>	<i>instance</i>	The number of the RTC instance used
<i>in</i>	<i>time</i>	Pointer to the variable in which the time is stored

**Returns**

STATUS\_SUCCESS if the operation was successful, STATUS\_ERROR if the time provided was invalid or if the counter was not stopped.

Definition at line 382 of file rtc\_driver.c.

**16.110.5.22** status\_t RTC\_DRV\_StartCounter ( uint32\_t *instance* )

Start RTC instance counter. Before calling this function the user should use RTC\_DRV\_SetTimeDate to configure the start time.

**Parameters**

<i>in</i>	<i>instance</i>	The number of the RTC instance used
-----------	-----------------	-------------------------------------

**Returns**

STATUS\_SUCCESS if the operation was successful, STATUS\_ERROR if the counter cannot be enabled or is already enabled.

Definition at line 265 of file rtc\_driver.c.

**16.110.5.23** status\_t RTC\_DRV\_StopCounter ( uint32\_t *instance* )

Disable RTC instance counter.



**Parameters**

<code>in</code>	<i>instance</i>	The number of the RTC instance used
-----------------	-----------------	-------------------------------------

**Returns**

STATUS\_SUCCESS if the operation was successful, STATUS\_ERROR if the counter could not be stopped.

Definition at line 296 of file rtc\_driver.c.

## 16.111 Raw API

### 16.111.1 Detailed Description

The raw API is operating on PDU level and it is typically used to gateway PDUs between CAN and LIN.

Usually, a FIFO is used to buffer PDUs in order to handle the different bus speeds.

#### Functions

- void [ld\\_put\\_raw](#) (I\_ifc\_handle iii, const I\_u8 \*const data)  
*Queue the transmission of 8 bytes of data in one frame.*
- void [ld\\_get\\_raw](#) (I\_ifc\_handle iii, I\_u8 \*const data)  
*Copy the oldest received diagnostic frame data to the memory specified by data.*
- I\_u8 [ld\\_raw\\_tx\\_status](#) (I\_ifc\_handle iii)  
*Get the status of the raw frame transmission function.*
- I\_u8 [ld\\_raw\\_rx\\_status](#) (I\_ifc\_handle iii)  
*Get the status of the raw frame receive function.*

### 16.111.2 Function Documentation

#### 16.111.2.1 void ld\_get\_raw ( I\_ifc\_handle iii, I\_u8 \*const data )

Copy the oldest received diagnostic frame data to the memory specified by data.

##### Parameters

in	iii	Interface name
in	data	Buffer for the data to be transmitted

##### Returns

void

Copy the oldest received diagnostic frame data to the memory specified by data. The data returned is received from master request frame for slave node and the slave response frame for master node.

Definition at line 161 of file lin\_commontl\_api.c.

#### 16.111.2.2 void ld\_put\_raw ( I\_ifc\_handle iii, const I\_u8 \*const data )

Queue the transmission of 8 bytes of data in one frame.

##### Parameters

in	iii	Interface name
in	data	Buffer for the data to be transmitted

##### Returns

void

Queue the transmission of 8 bytes of data in one frame The data is sent in the next suitable frame.

Definition at line 129 of file lin\_commontl\_api.c.

#### 16.111.2.3 I\_u8 ld\_raw\_rx\_status ( I\_ifc\_handle iii )

Get the status of the raw frame receive function.

**Parameters**

<i>in</i>	<i>iii</i>	Interface name
-----------	------------	----------------

**Returns**

*I\_u8*

Get the status of the raw frame receive function: LD\_NO\_DATA The receive queue is empty.(For LIN2.1 and above only) LD\_DATA\_AVAILABLE The receive queue contains data that can be read. LD\_RECEIVE\_ERROR LIN protocol errors occurred during the transfer; initialize and redo the transfer.(For LIN2.1 and above only). LD\_TRANSFER\_ERROR: (For LIN2.0 and J2602 only) LIN protocol errors occurred during the transfer; initialize and redo the transfer.

Definition at line 193 of file lin\_commontl\_api.c.

**16.111.2.4 *I\_u8* ld\_raw\_tx\_status ( *I\_ifc\_handle iii* )**

Get the status of the raw frame transmission function.

**Parameters**

<i>in</i>	<i>iii</i>	Interface name
-----------	------------	----------------

**Returns**

*I\_u8*

Get the status of the raw frame transmission function: This function is available for < br / > LD\_QUEUE\_EMPTY : The transmit queue is empty. In case previous calls to < br / > ld\_put\_raw, all frames in the queue have been < br / > transmitted. < br / > LD\_QUEUE\_AVAILABLE: The transmit queue contains entries, but is not full. < br / > (For LIN2.1 and above only). LD\_QUEUE\_FULL : The transmit queue is full and can not accept further < br / > frames. < br / > LD\_TRANSMIT\_ERROR : (For LIN2.1 and above only) LIN protocol errors occurred during the transfer; initialize and redo the transfer. LD\_TRANSFER\_ERROR: (For LIN2.0 and J2602 only) LIN protocol errors occurred during the transfer; initialize and redo the transfer.

Definition at line 178 of file lin\_commontl\_api.c.

## 16.112 Real Time Clock Driver (RTC)

### 16.112.1 Detailed Description

The S32 SDK provides the Peripheral Driver for the Real Time Clock (RTC) module of S32 SDK devices.

#### Hardware background

The Real Time Clock Module is a independent timer that keeps track of the exact date and time with no software overhead, with low power usage.

Features of the RTC module include:

- 32-bit seconds counter with roll-over protection and 32-bit alarm
- 16-bit prescaler with compensation that can correct errors between 0.12 ppm and 3906 ppm
- Option to increment prescaler using the LPO (prescaler increments by 32 every clock edge)
- Register write protection
- Lock register requires POR or software reset to enable write access
- Configurable 1, 2, 4, 8, 16, 32, 64 or 128 Hz square wave output with optional interrupt
- Alarm interrupt configured by the driver automatically refreshes alarm time configured by the user
- User interrupt handlers can be configured for all interrupts

#### How to use the RTC driver in your application

In order to be able to use the RTC in your application, the first thing to do is initializing it with the desired configuration. This is done by calling the **RTC\_DRV\_Init** function. One of the arguments passed to this function is the configuration which will be used for the RTC instance, specified by the **rtc\_init\_config\_t** structure.

The **rtc\_init\_config\_t** structure allows you to configure the following:

- RTC clock source (32 KHz clock or 1 KHz LPO clock)
- Clock Out pin configuration (Clock OUT pin source)
- Compensation (Interval and value)
- Update enable - this allows updates to Time Counter Enable bit if the Status Register under limited conditions
- Enable non supervisor writes to the registers

The **rtc\_seconds\_int\_config\_t** structure configures the **time seconds interrupt**. To setup an interrupt every seconds you have to configure the structure mentioned with the following parameters:

- Frequency of the interrupt
- Interrupt Handler
- If needed - interrupt handler parameters

An alarm is configured with **rtc\_alarm\_config\_t** structure, which is described by the following parameters:

- Alarm time in date-time format
- Interval of alarm repeat in seconds
- Number of alarm repeats (use 0 if the alarm is not recursive)
- Repeat forever field (if set, the number of repeats field will be ignored)

- Alarm interrupt enable
- Alarm interrupt handler
- Alarm interrupt handler parameters

#### Note

If the alarm interrupt is not enabled, the user must make the updates of the alarm time manually.

After the `RTC_DRV_Init()` function call and, if needed, alarm and other configurations the RTC counter is started by calling `RTC_DRV_StartCounter()`.

To update desired time date use `RTC_DRV_SetTimeDate()` function, this method uses a Time and Date structure `rtc_timedate_t` in a calendar format mode.

To get the current time and date you can call `RTC_DRV_GetCurrentTimeDate()` function, this method will get the seconds from the Time Seconds Register and will convert into human readable format as `rtc_timedate_t`.

To check if a structure `rtc_timedate_t` is properly configured use `RTC_DRV_IsTimeDateCorrectFormat()` function that will return true if configuration is valid or false if configuration is invalid.

To set an alarm at a desired date and time use `RTC_DRV_ConfigureAlarm()` function, this method uses a structure `rtc_alarm_config_t` with Time, Date and Alarm Handler and will trigger at set time an interrupt set by user.

To get the configured alarm use `RTC_DRV_GetAlarmConfig()` function, this method will return the Time and Date for alarm in a structure `rtc_alarm_config_t`.

To check if an alarm is pending use `RTC_DRV_IsAlarmPending()` function, this method will return true if alarm is pending or false if no alarm pending.

To configure a seconds interrupt use `RTC_DRV_ConfigureSecondsInt()` function, this method use structure `rtc_↵_seconds_int_config_t` to be configured with a callback function.

After driver configuration the user can use `RTC_DRV_StartCounter()` function to start the timer and `RTC_DRV_↵StopCounter()` function to stop it.

To lock access to RTC registers use `RTC_DRV_ConfigureRegisterLock()` function, this method uses a structure `rtc_register_lock_config_t` that describes what registers will be locked. Attention all the registers are unlocked only by software reset or power on reset.

To check if RTC registers are locked use `RTC_DRV_GetRegisterLock()` function, this will return a structure `rtc_↵register_lock_config_t` with locked registers.

To convert seconds to a human readable value use `RTC_DRV_ConvertSecondsToTimeDate()` function, this will return a structure `rtc_timedate_t` based on the seconds value.

To convert a time date to seconds use `RTC_DRV_ConvertTimeDateToSeconds()` function, this will return seconds value based on time date structure `rtc_timedate_t`.

To get the time date for next alarm use `RTC_DRV_GetNextAlarmTime()` function, this will return a structure `rtc_↵timedate_t`.

To configure a fault handler for cases as Overflow and Invalid Time use `RTC_DRV_ConfigureFaultInt()` function, this method will use a structure `rtc_interrupt_config_t` with a callback function.

#### Example

```
/* Time Seconds interrupt handler */
void secondsISR(void)
{
    /* Do Something */
}

void rtcAlarmCallback(void)
{
    rtc_timedate_t currentTime;
    RTC_DRV_GetCurrentTimeDate(0U, &currentTime);

    /* Do something with the time and date */
}
```

```

}

int main()
{
    rtc_seconds_int_config_t rtcTimer1_SecIntConfig0 =
    {
        .secondIntConfig          =    RTC_INT_1HZ,
        .secondIntEnable          =    true,
        .rtcSecondsCallback        =    secondsISR,
        .secondsCallbackParams     =    NULL
    };

    /* rtcTimer1 configuration structure */
    const rtc_init_config_t rtcTimer1_Config0 =
    {
        /* Time compensation interval */
        .compensationInterval      =    0U,
        /* Time compensation value */
        .compensation              =    0,
        /* RTC Clock Source is 32 KHz crystal */
        .clockSelect               =    RTC_CLK_SRC_OSC_32KHZ,
        /* RTC Clock Out is 32 KHz clock */
        .clockOutConfig            =    RTC_CLKOUT_SRC_32KHZ,
        /* Update of the TCE bit is not allowed */
        .updateEnable              =    false,
        /* Non-supervisor mode write accesses are not supported and generate
         * a bus error.
         */
        .nonSupervisorAccessEnable =    false
    };

    /* RTC Initial Time and Date */
    rtc_timedate_t                rtcStartTime =
    {
        /* Year */
        .year                     =    2016U,
        /* Month */
        .month                    =    01U,
        /* Day */
        .day                      =    01U,
        /* Hour */
        .hour                     =    00U,
        /* Minutes */
        .minutes                  =    00U,
        /* Seconds */
        .seconds                  =    00U
    };

    /* rtcTimer1 Alarm configuration 0 */
    rtc_alarm_config_t            alarmConfig0 =
    {
        /* Alarm Date */
        .alarmTime                =
        {
            /* Year */
            .year                 =    2016U,
            /* Month */
            .month                =    01U,
            /* Day */
            .day                  =    01U,
            /* Hour */
            .hour                 =    00U,
            /* Minutes */
            .minutes              =    00U,
            /* Seconds */
            .seconds              =    03U,
        },

        /* Alarm repeat interval */
        .repetitionInterval       =    3UL,

        /* Number of alarm repeats */
        .numberOfRepeats          =    0UL,

        /* Repeat alarm forever */
        .repeatForever            =    true,

        /* Alarm interrupt disabled */
        .alarmIntEnable           =    true,

        /* Alarm interrupt handler */
        .alarmCallback            =    (void *)rtcAlarmCallback,

        /* Alarm interrupt handler parameters */
        .callbackParams           =    (void *)NULL
    };
}

```

```

/* Call the init function */
RTC_DRV_Init(0UL, &rtcInitConfig);

/* Set the time and date */
RTC_DRV_SetTimeDate(0UL, &rtcStartTime);

/* Configure RTC Time Seconds Interrupt */
RTC_DRV_ConfigureSecondsInt(0UL, &rtcTimer1_SecIntConfig0);

/* Start RTC counter */
RTC_DRV_StartCounter(0UL);

/* Configure an alarm every 3 seconds */
RTC_DRV_ConfigureAlarm(0UL, &rtcAlarmConfig0);

while(1);
}

```

### Important Notes

- Before using the RTC driver the module clock must be configured

#### Note

When using the on chip LPO clock as source input for the RTC, the user needs to make sure that the LPO generates the desired frequency by adjusting the LPO trimming value.

For more details about LPO trimming please consult the available documentation.

- The driver enables the interrupts for the corresponding RTC module, but any interrupt priority must be done by the application
- The board specific configurations must be done prior to driver calls; the driver has no influence on the functionality of the clockout pin - they must be configured by application
- If Non-supervisor mode write accesses are supported you need to set AIPS to allow usermode access to RTC Memory Space

### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```

${S32SDK_PATH}\platform\drivers\src\rtc\rtc_driver.c
${S32SDK_PATH}\platform\drivers\src\rtc\rtc_hw_access.c
${S32SDK_PATH}\platform\drivers\src\rtc\rtc_irq.c

```

#### Include path

The following paths need to be added to the include path of the toolchain:

```

${S32SDK_PATH}\platform\drivers\inc\

```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

1. [Clock Manager](#)
2. [Interrupt Manager \(Interrupt\)](#)

#### Modules

- [RTC Driver](#)

*Real Time Clock Peripheral Driver.*

## 16.113 Register address map

### 16.113.1 Detailed Description

The SPI protocol requires the first byte transmitted on MOSI line to be set to register address the user wants to access (write/read). The byte consists of ADDRESS[7:1] and READ\_BIT[0]. This section lists all the register addresses available on UJA113x devices which can be used throughout the source code instead of fixed values. The defined addresses are already shifted.

The address is useful in [UJA113X\\_SbcTransfer\(\)](#) function, which allows generic read/write access.

#### Primary control registers (0x00 - 0x0F)

- #define [WTDC\\_ADDRESS](#) ((uint8\_t) 0x00U << 1U)  
*Watchdog control register address.*
- #define [MC\\_ADDRESS](#) ((uint8\_t) 0x01U << 1U)  
*Mode control register address.*
- #define [FSC\\_ADDRESS](#) ((uint8\_t) 0x02U << 1U)  
*Fail-safe control register address.*
- #define [MSBCS\\_ADDRESS](#) ((uint8\_t) 0x03U << 1U)  
*Main SBC status register address.*
- #define [SYSIE\\_ADDRESS](#) ((uint8\_t) 0x04U << 1U)  
*System interrupt enable register address.*
- #define [WTDS\\_ADDRESS](#) ((uint8\_t) 0x05U << 1U)  
*Watchdog status register address.*
- #define [MEM0\\_ADDRESS](#) ((uint8\_t) 0x06U << 1U)  
*General purpose memory 0 address.*
- #define [MEM1\\_ADDRESS](#) ((uint8\_t) 0x07U << 1U)  
*General purpose memory 1 address.*
- #define [MEM2\\_ADDRESS](#) ((uint8\_t) 0x08U << 1U)  
*General purpose memory 2 address.*
- #define [MEM3\\_ADDRESS](#) ((uint8\_t) 0x09U << 1U)  
*General purpose memory 3 address.*
- #define [LCKC\\_ADDRESS](#) ((uint8\_t) 0x0AU << 1U)  
*Lock control register address.*

#### Supply control registers (0x10 - 0x1F)

- #define [RC\\_ADDRESS](#) ((uint8\_t) 0x10U << 1U)  
*Regulator control register address.*
- #define [BMETSC\\_ADDRESS](#) ((uint8\_t) 0x11U << 1U)  
*Battery monitor event trigger source control register address.*
- #define [BMUTC\\_ADDRESS](#) ((uint8\_t) 0x12U << 1U)  
*Battery monitor undervoltage threshold control register address.*
- #define [BMOTC\\_ADDRESS](#) ((uint8\_t) 0x13U << 1U)  
*Battery monitor overvoltage threshold control register address.*
- #define [BMHC\\_ADDRESS](#) ((uint8\_t) 0x14U << 1U)  
*Battery monitor hysteresis control register address.*
- #define [VBADC1\\_ADDRESS](#) ((uint8\_t) 0x15U << 1U)  
*ADC conversion results for Vbat register 1 address.*
- #define [VBADC2\\_ADDRESS](#) ((uint8\_t) 0x16U << 1U)  
*ADC conversion results for Vbat register 2 address.*



- #define **BSADC1\_ADDRESS** ((uint8\_t) 0x17U << 1U)  
*ADC conversion results for Vbatsense register 1 address.*
- #define **BSADC2\_ADDRESS** ((uint8\_t) 0x18U << 1U)  
*ADC conversion results for Vbatsense register 2 address.*
- #define **SMPSC\_ADDRESS** ((uint8\_t) 0x19U << 1U)  
*SMPS control register address.*
- #define **SMPSOVC\_ADDRESS** ((uint8\_t) 0x1AU << 1U)  
*SMPS out voltage control register address.*
- #define **SUPVS\_ADDRESS** ((uint8\_t) 0x1BU << 1U)  
*Supply voltage status register address.*
- #define **SUPIE\_ADDRESS** ((uint8\_t) 0x1CU << 1U)  
*Supply interrupt enable register address.*

#### Transceiver control registers (0x20 - 0x2F)

- #define **CANC\_ADDRESS** ((uint8\_t) 0x20U << 1U)  
*CAN control register address.*
- #define **LINC\_ADDRESS** ((uint8\_t) 0x21U << 1U)  
*LIN control register address.*
- #define **TS\_ADDRESS** ((uint8\_t) 0x22U << 1U)  
*Transceiver status register address.*
- #define **TIE\_ADDRESS** ((uint8\_t) 0x23U << 1U)  
*Transceiver interrupt enable register address.*
- #define **DR\_ADDRESS** ((uint8\_t) 0x26U << 1U)  
*Data rate register address.*
- #define **ID0\_ADDRESS** ((uint8\_t) 0x27U << 1U)  
*ID register 0 address.*
- #define **ID1\_ADDRESS** ((uint8\_t) 0x28U << 1U)  
*ID register 1 address.*
- #define **ID2\_ADDRESS** ((uint8\_t) 0x29U << 1U)  
*ID register 2 address.*
- #define **ID3\_ADDRESS** ((uint8\_t) 0x2AU << 1U)  
*ID register 3 address.*
- #define **IDM0\_ADDRESS** ((uint8\_t) 0x2BU << 1U)  
*ID mask register 0 address.*
- #define **IDM1\_ADDRESS** ((uint8\_t) 0x2CU << 1U)  
*ID mask register 1 address.*
- #define **IDM2\_ADDRESS** ((uint8\_t) 0x2DU << 1U)  
*ID mask register 2 address.*
- #define **IDM3\_ADDRESS** ((uint8\_t) 0x2EU << 1U)  
*ID mask register 3 address.*
- #define **FC\_ADDRESS** ((uint8\_t) 0x2FU << 1U)  
*Frame control register address.*

## HVIO control registers (0x30 - 0x4F)

- #define [HVIO1C\\_ADDRESS](#) ((uint8\_t) 0x30U << 1U)  
*HVIO 1 control register address.*
- #define [HVIO2C\\_ADDRESS](#) ((uint8\_t) 0x31U << 1U)  
*HVIO 2 control register address.*
- #define [HVIO3C\\_ADDRESS](#) ((uint8\_t) 0x32U << 1U)  
*HVIO 3 control register address.*
- #define [HVIO4C\\_ADDRESS](#) ((uint8\_t) 0x33U << 1U)  
*HVIO 4 control register address.*
- #define [B0WUPTHRC\\_ADDRESS](#) ((uint8\_t) 0x34U << 1U)  
*Bank 0 (HVIO 1 - 4) wake-up threshold control register address.*
- #define [B0WUPS\\_ADDRESS](#) ((uint8\_t) 0x35U << 1U)  
*Bank 0 wake-up status register address.*
- #define [B0DRVS\\_ADDRESS](#) ((uint8\_t) 0x36U << 1U)  
*Bank 0 driver status register address.*
- #define [B0WUPIE\\_ADDRESS](#) ((uint8\_t) 0x37U << 1U)  
*Bank 0 wake-up interrupt enable register address.*
- #define [B0FIE\\_ADDRESS](#) ((uint8\_t) 0x38U << 1U)  
*Bank 0 fail interrupt enable register address.*
- #define [B0SCDTC\\_ADDRESS](#) ((uint8\_t) 0x39U << 1U)  
*Bank 0 short-circuit detection threshold control register address.*
- #define [B0OLDTC\\_ADDRESS](#) ((uint8\_t) 0x3AU << 1U)  
*Bank 0 open-laod detection threshold control register address.*
- #define [HVIO5C\\_ADDRESS](#) ((uint8\_t) 0x40U << 1U)  
*HVIO 5 control register address.*
- #define [HVIO6C\\_ADDRESS](#) ((uint8\_t) 0x41U << 1U)  
*HVIO 6 control register address.*
- #define [HVIO7C\\_ADDRESS](#) ((uint8\_t) 0x42U << 1U)  
*HVIO 7 control register address.*
- #define [HVIO8C\\_ADDRESS](#) ((uint8\_t) 0x43U << 1U)  
*HVIO 8 control register address.*
- #define [B1WUPTHRC\\_ADDRESS](#) ((uint8\_t) 0x44U << 1U)  
*Bank 1 (HVIO 5 - 8) wake-up threshold control register address.*
- #define [B1WUPS\\_ADDRESS](#) ((uint8\_t) 0x45U << 1U)  
*Bank 1 wake-up status register address.*
- #define [B1DRVS\\_ADDRESS](#) ((uint8\_t) 0x46U << 1U)  
*Bank 1 driver status register address.*
- #define [B1WUPIE\\_ADDRESS](#) ((uint8\_t) 0x47U << 1U)  
*Bank 1 wake-up interrupt enable register address.*
- #define [B1FIE\\_ADDRESS](#) ((uint8\_t) 0x48U << 1U)  
*Bank 1 fail interrupt enable register address.*
- #define [B1SCDTC\\_ADDRESS](#) ((uint8\_t) 0x49U << 1U)  
*Bank 1 short-circuit detection threshold control register address.*
- #define [B1OLDTC\\_ADDRESS](#) ((uint8\_t) 0x4AU << 1U)  
*Bank 1 open-laod detection threshold control register address.*

## Timer control registers (0x50 - 0x5F)

- #define **TIM1C\_ADDRESS** ((uint8\_t) 0x50U << 1U)  
*Timer 1 control register address.*
- #define **TIM1DCC\_ADDRESS** ((uint8\_t) 0x51U << 1U)  
*Timer 1 duty cycle control register address.*
- #define **TIM2C\_ADDRESS** ((uint8\_t) 0x52U << 1U)  
*Timer 2 control register address.*
- #define **TIM2DCC\_ADDRESS** ((uint8\_t) 0x53U << 1U)  
*Timer 2 duty cycle control register address.*
- #define **TIM3C\_ADDRESS** ((uint8\_t) 0x54U << 1U)  
*Timer 3 control register address.*
- #define **TIM3DCC\_ADDRESS** ((uint8\_t) 0x55U << 1U)  
*Timer 3 duty cycle control register address.*
- #define **TIM4C\_ADDRESS** ((uint8\_t) 0x56U << 1U)  
*Timer 4 control register address.*
- #define **TIM4DCC\_ADDRESS** ((uint8\_t) 0x57U << 1U)  
*Timer 4 duty cycle control register address.*

## Interrupt status registers (0x60 - 0x6F)

- #define **GIS\_ADDRESS** ((uint8\_t) 0x60U << 1U)  
*Global interrupt status register address.*
- #define **SYSIS\_ADDRESS** ((uint8\_t) 0x61U << 1U)  
*System interrupt status register address.*
- #define **SUPIS\_ADDRESS** ((uint8\_t) 0x62U << 1U)  
*Supply interrupt status register address.*
- #define **TIS\_ADDRESS** ((uint8\_t) 0x63U << 1U)  
*Transceiver interrupt status register address.*
- #define **B0WUPIS\_ADDRESS** ((uint8\_t) 0x64U << 1U)  
*Bank 0 wake-up interrupt status register address.*
- #define **B0FIS\_ADDRESS** ((uint8\_t) 0x65U << 1U)  
*Bank 0 fail interrupt status register address.*
- #define **B1WUPIS\_ADDRESS** ((uint8\_t) 0x66U << 1U)  
*Bank 1 wake-up interrupt status register address.*
- #define **B1FIS\_ADDRESS** ((uint8\_t) 0x67U << 1U)  
*Bank 1 fail interrupt status register address.*
- #define **DM0\_ADDRESS** ((uint8\_t) 0x68U << 1U)  
*Data mask register 0 address.*
- #define **DM1\_ADDRESS** ((uint8\_t) 0x69U << 1U)  
*Data mask register 1 address.*
- #define **DM2\_ADDRESS** ((uint8\_t) 0x6AU << 1U)  
*Data mask register 2 address.*
- #define **DM3\_ADDRESS** ((uint8\_t) 0x6BU << 1U)  
*Data mask register 3 address.*
- #define **DM4\_ADDRESS** ((uint8\_t) 0x6CU << 1U)  
*Data mask register 4 address.*
- #define **DM5\_ADDRESS** ((uint8\_t) 0x6DU << 1U)  
*Data mask register 5 address.*
- #define **DM6\_ADDRESS** ((uint8\_t) 0x6EU << 1U)  
*Data mask register 6 address.*
- #define **DM7\_ADDRESS** ((uint8\_t) 0x6FU << 1U)  
*Data mask register 7 address.*

## MTPNV and ID registers (0x70 - 0x7F)

- #define [MTPNVS\\_ADDRESS](#) ((uint8\_t) 0x70U << 1U)  
*MTPNV status register address.*
- #define [HVIOHSDRVC\\_ADDRESS](#) ((uint8\_t) 0x71U << 1U)  
*HVIO high-side driver control register address.*
- #define [HVIOLOSDRVC\\_ADDRESS](#) ((uint8\_t) 0x72U << 1U)  
*HVIO low-side driver control register address.*
- #define [SUPC\\_ADDRESS](#) ((uint8\_t) 0x73U << 1U)  
*Start-up control register address.*
- #define [SBCCC\\_ADDRESS](#) ((uint8\_t) 0x74U << 1U)  
*SBC configuration control register address.*
- #define [MTPNVCRCC\\_ADDRESS](#) ((uint8\_t) 0x75U << 1U)  
*MTPNV CRC control register address.*
- #define [DID1\\_ADDRESS](#) ((uint8\_t) 0x7EU << 1U)  
*Device identification register 1 address.*
- #define [DID2\\_ADDRESS](#) ((uint8\_t) 0x7FU << 1U)  
*Device identification register 2 address.*

## 16.113.2 Macro Definition Documentation

### 16.113.2.1 #define B0DRVS\_ADDRESS ((uint8\_t) 0x36U << 1U)

Bank 0 driver status register address.

Definition at line 145 of file sbc\_uja113x.h.

### 16.113.2.2 #define B0FIE\_ADDRESS ((uint8\_t) 0x38U << 1U)

Bank 0 fail interrupt enable register address.

Definition at line 147 of file sbc\_uja113x.h.

### 16.113.2.3 #define B0FIS\_ADDRESS ((uint8\_t) 0x65U << 1U)

Bank 0 fail interrupt status register address.

Definition at line 186 of file sbc\_uja113x.h.

### 16.113.2.4 #define B0OLDTC\_ADDRESS ((uint8\_t) 0x3AU << 1U)

Bank 0 open-laod detection threshold control register address.

Definition at line 149 of file sbc\_uja113x.h.

### 16.113.2.5 #define B0SCDTC\_ADDRESS ((uint8\_t) 0x39U << 1U)

Bank 0 short-circuit detection threshold control register address.

Definition at line 148 of file sbc\_uja113x.h.

### 16.113.2.6 #define B0WUPIE\_ADDRESS ((uint8\_t) 0x37U << 1U)

Bank 0 wake-up interrupt enable register address.

Definition at line 146 of file sbc\_uja113x.h.

### 16.113.2.7 #define B0WUPIS\_ADDRESS ((uint8\_t) 0x64U << 1U)

Bank 0 wake-up interrupt status register address.

Definition at line 185 of file sbc\_uja113x.h.

**16.113.2.8** #define B0WUPS\_ADDRESS ((uint8\_t) 0x35U << 1U)

Bank 0 wake-up status register address.

Definition at line 144 of file sbc\_uja113x.h.

**16.113.2.9** #define B0WUPTHRC\_ADDRESS ((uint8\_t) 0x34U << 1U)

Bank 0 (HVIO 1 - 4) wake-up threshold control register address.

Definition at line 143 of file sbc\_uja113x.h.

**16.113.2.10** #define B1DRVS\_ADDRESS ((uint8\_t) 0x46U << 1U)

Bank 1 driver status register address.

Definition at line 156 of file sbc\_uja113x.h.

**16.113.2.11** #define B1FIE\_ADDRESS ((uint8\_t) 0x48U << 1U)

Bank 1 fail interrupt enable register address.

Definition at line 158 of file sbc\_uja113x.h.

**16.113.2.12** #define B1FIS\_ADDRESS ((uint8\_t) 0x67U << 1U)

Bank 1 fail interrupt status register address.

Definition at line 188 of file sbc\_uja113x.h.

**16.113.2.13** #define B1OLDTC\_ADDRESS ((uint8\_t) 0x4AU << 1U)

Bank 1 open-load detection threshold control register address.

Definition at line 160 of file sbc\_uja113x.h.

**16.113.2.14** #define B1SCDTC\_ADDRESS ((uint8\_t) 0x49U << 1U)

Bank 1 short-circuit detection threshold control register address.

Definition at line 159 of file sbc\_uja113x.h.

**16.113.2.15** #define B1WUPIE\_ADDRESS ((uint8\_t) 0x47U << 1U)

Bank 1 wake-up interrupt enable register address.

Definition at line 157 of file sbc\_uja113x.h.

**16.113.2.16** #define B1WUPIS\_ADDRESS ((uint8\_t) 0x66U << 1U)

Bank 1 wake-up interrupt status register address.

Definition at line 187 of file sbc\_uja113x.h.

**16.113.2.17** #define B1WUPS\_ADDRESS ((uint8\_t) 0x45U << 1U)

Bank 1 wake-up status register address.

Definition at line 155 of file sbc\_uja113x.h.

**16.113.2.18** #define B1WUPTHRC\_ADDRESS ((uint8\_t) 0x44U << 1U)

Bank 1 (HVIO 5 - 8) wake-up threshold control register address.

Definition at line 154 of file sbc\_uja113x.h.

16.113.2.19 `#define BMETSC_ADDRESS ((uint8_t) 0x11U << 1U)`

Battery monitor event trigger source control register address.

Definition at line 101 of file sbc\_uja113x.h.

16.113.2.20 `#define BMHC_ADDRESS ((uint8_t) 0x14U << 1U)`

Battery monitor hysteresis control register address.

Definition at line 104 of file sbc\_uja113x.h.

16.113.2.21 `#define BMOTC_ADDRESS ((uint8_t) 0x13U << 1U)`

Battery monitor overvoltage threshold control register address.

Definition at line 103 of file sbc\_uja113x.h.

16.113.2.22 `#define BMUTC_ADDRESS ((uint8_t) 0x12U << 1U)`

Battery monitor undervoltage threshold control register address.

Definition at line 102 of file sbc\_uja113x.h.

16.113.2.23 `#define BSADC1_ADDRESS ((uint8_t) 0x17U << 1U)`

ADC conversion results for Vbatsense register 1 address.

Definition at line 107 of file sbc\_uja113x.h.

16.113.2.24 `#define BSADC2_ADDRESS ((uint8_t) 0x18U << 1U)`

ADC conversion results for Vbatsense register 2 address.

Definition at line 108 of file sbc\_uja113x.h.

16.113.2.25 `#define CANC_ADDRESS ((uint8_t) 0x20U << 1U)`

CAN control register address.

Definition at line 119 of file sbc\_uja113x.h.

16.113.2.26 `#define DID1_ADDRESS ((uint8_t) 0x7EU << 1U)`

Device identification register 1 address.

Definition at line 209 of file sbc\_uja113x.h.

16.113.2.27 `#define DID2_ADDRESS ((uint8_t) 0x7FU << 1U)`

Device identification register 2 address.

Definition at line 210 of file sbc\_uja113x.h.

16.113.2.28 `#define DM0_ADDRESS ((uint8_t) 0x68U << 1U)`

Data mask register 0 address.

Definition at line 189 of file sbc\_uja113x.h.

16.113.2.29 `#define DM1_ADDRESS ((uint8_t) 0x69U << 1U)`

Data mask register 1 address.

Definition at line 190 of file sbc\_uja113x.h.

**16.113.2.30** `#define DM2_ADDRESS ((uint8_t) 0x6AU << 1U)`

Data mask register 2 address.

Definition at line 191 of file sbc\_uja113x.h.

**16.113.2.31** `#define DM3_ADDRESS ((uint8_t) 0x6BU << 1U)`

Data mask register 3 address.

Definition at line 192 of file sbc\_uja113x.h.

**16.113.2.32** `#define DM4_ADDRESS ((uint8_t) 0x6CU << 1U)`

Data mask register 4 address.

Definition at line 193 of file sbc\_uja113x.h.

**16.113.2.33** `#define DM5_ADDRESS ((uint8_t) 0x6DU << 1U)`

Data mask register 5 address.

Definition at line 194 of file sbc\_uja113x.h.

**16.113.2.34** `#define DM6_ADDRESS ((uint8_t) 0x6EU << 1U)`

Data mask register 6 address.

Definition at line 195 of file sbc\_uja113x.h.

**16.113.2.35** `#define DM7_ADDRESS ((uint8_t) 0x6FU << 1U)`

Data mask register 7 address.

Definition at line 196 of file sbc\_uja113x.h.

**16.113.2.36** `#define DR_ADDRESS ((uint8_t) 0x26U << 1U)`

Data rate register address.

Definition at line 123 of file sbc\_uja113x.h.

**16.113.2.37** `#define FC_ADDRESS ((uint8_t) 0x2FU << 1U)`

Frame control register address.

Definition at line 132 of file sbc\_uja113x.h.

**16.113.2.38** `#define FSC_ADDRESS ((uint8_t) 0x02U << 1U)`

Fail-safe control register address.

Definition at line 85 of file sbc\_uja113x.h.

**16.113.2.39** `#define GIS_ADDRESS ((uint8_t) 0x60U << 1U)`

Global interrupt status register address.

Definition at line 181 of file sbc\_uja113x.h.

**16.113.2.40** `#define HVIO1C_ADDRESS ((uint8_t) 0x30U << 1U)`

HVIO 1 control register address.

Definition at line 139 of file sbc\_uja113x.h.

**16.113.2.41 #define HVIO2C\_ADDRESS ((uint8\_t) 0x31U << 1U)**

HVIO 2 control register address.

Definition at line 140 of file sbc\_uja113x.h.

**16.113.2.42 #define HVIO3C\_ADDRESS ((uint8\_t) 0x32U << 1U)**

HVIO 3 control register address.

Definition at line 141 of file sbc\_uja113x.h.

**16.113.2.43 #define HVIO4C\_ADDRESS ((uint8\_t) 0x33U << 1U)**

HVIO 4 control register address.

Definition at line 142 of file sbc\_uja113x.h.

**16.113.2.44 #define HVIO5C\_ADDRESS ((uint8\_t) 0x40U << 1U)**

HVIO 5 control register address.

Definition at line 150 of file sbc\_uja113x.h.

**16.113.2.45 #define HVIO6C\_ADDRESS ((uint8\_t) 0x41U << 1U)**

HVIO 6 control register address.

Definition at line 151 of file sbc\_uja113x.h.

**16.113.2.46 #define HVIO7C\_ADDRESS ((uint8\_t) 0x42U << 1U)**

HVIO 7 control register address.

Definition at line 152 of file sbc\_uja113x.h.

**16.113.2.47 #define HVIO8C\_ADDRESS ((uint8\_t) 0x43U << 1U)**

HVIO 8 control register address.

Definition at line 153 of file sbc\_uja113x.h.

**16.113.2.48 #define HVIOHSDRVC\_ADDRESS ((uint8\_t) 0x71U << 1U)**

HVIO high-side driver control register address.

Definition at line 204 of file sbc\_uja113x.h.

**16.113.2.49 #define HVIOLOSDRVC\_ADDRESS ((uint8\_t) 0x72U << 1U)**

HVIO low-side driver control register address.

Definition at line 205 of file sbc\_uja113x.h.

**16.113.2.50 #define ID0\_ADDRESS ((uint8\_t) 0x27U << 1U)**

ID register 0 address.

Definition at line 124 of file sbc\_uja113x.h.

**16.113.2.51 #define ID1\_ADDRESS ((uint8\_t) 0x28U << 1U)**

ID register 1 address.

Definition at line 125 of file sbc\_uja113x.h.



16.113.2.52 `#define ID2_ADDRESS ((uint8_t) 0x29U << 1U)`

ID register 2 address.

Definition at line 126 of file sbc\_uja113x.h.

16.113.2.53 `#define ID3_ADDRESS ((uint8_t) 0x2AU << 1U)`

ID register 3 address.

Definition at line 127 of file sbc\_uja113x.h.

16.113.2.54 `#define IDM0_ADDRESS ((uint8_t) 0x2BU << 1U)`

ID mask register 0 address.

Definition at line 128 of file sbc\_uja113x.h.

16.113.2.55 `#define IDM1_ADDRESS ((uint8_t) 0x2CU << 1U)`

ID mask register 1 address.

Definition at line 129 of file sbc\_uja113x.h.

16.113.2.56 `#define IDM2_ADDRESS ((uint8_t) 0x2DU << 1U)`

ID mask register 2 address.

Definition at line 130 of file sbc\_uja113x.h.

16.113.2.57 `#define IDM3_ADDRESS ((uint8_t) 0x2EU << 1U)`

ID mask register 3 address.

Definition at line 131 of file sbc\_uja113x.h.

16.113.2.58 `#define LCKC_ADDRESS ((uint8_t) 0x0AU << 1U)`

Lock control register address.

Definition at line 93 of file sbc\_uja113x.h.

16.113.2.59 `#define LINC_ADDRESS ((uint8_t) 0x21U << 1U)`

LIN control register address.

Definition at line 120 of file sbc\_uja113x.h.

16.113.2.60 `#define MC_ADDRESS ((uint8_t) 0x01U << 1U)`

Mode control register address.

Definition at line 84 of file sbc\_uja113x.h.

16.113.2.61 `#define MEM0_ADDRESS ((uint8_t) 0x06U << 1U)`

General purpose memory 0 address.

Definition at line 89 of file sbc\_uja113x.h.

16.113.2.62 `#define MEM1_ADDRESS ((uint8_t) 0x07U << 1U)`

General purpose memory 1 address.

Definition at line 90 of file sbc\_uja113x.h.

**16.113.2.63** `#define MEM2_ADDRESS ((uint8_t) 0x08U << 1U)`

General purpose memory 2 address.

Definition at line 91 of file sbc\_uja113x.h.

**16.113.2.64** `#define MEM3_ADDRESS ((uint8_t) 0x09U << 1U)`

General purpose memory 3 address.

Definition at line 92 of file sbc\_uja113x.h.

**16.113.2.65** `#define MSBCS_ADDRESS ((uint8_t) 0x03U << 1U)`

Main SBC status register address.

Definition at line 86 of file sbc\_uja113x.h.

**16.113.2.66** `#define MTPNVCRC_ADDRESS ((uint8_t) 0x75U << 1U)`

MTPVN CRC control register address.

Definition at line 208 of file sbc\_uja113x.h.

**16.113.2.67** `#define MTPNVS_ADDRESS ((uint8_t) 0x70U << 1U)`

MTPNV status register address.

Definition at line 203 of file sbc\_uja113x.h.

**16.113.2.68** `#define RC_ADDRESS ((uint8_t) 0x10U << 1U)`

Regulator control register address.

Definition at line 100 of file sbc\_uja113x.h.

**16.113.2.69** `#define SBCCC_ADDRESS ((uint8_t) 0x74U << 1U)`

SBC configuration control register address.

Definition at line 207 of file sbc\_uja113x.h.

**16.113.2.70** `#define SMPSC_ADDRESS ((uint8_t) 0x19U << 1U)`

SMPS control register address.

Definition at line 109 of file sbc\_uja113x.h.

**16.113.2.71** `#define SMPSOVC_ADDRESS ((uint8_t) 0x1AU << 1U)`

SMPS out voltage control register address.

Definition at line 110 of file sbc\_uja113x.h.

**16.113.2.72** `#define SUPC_ADDRESS ((uint8_t) 0x73U << 1U)`

Start-up control register address.

Definition at line 206 of file sbc\_uja113x.h.

**16.113.2.73** `#define SUPIE_ADDRESS ((uint8_t) 0x1CU << 1U)`

Supply interrupt enable register address.

Definition at line 112 of file sbc\_uja113x.h.

16.113.2.74 `#define SUPIS_ADDRESS ((uint8_t) 0x62U << 1U)`

Supply interrupt status register address.

Definition at line 183 of file sbc\_uja113x.h.

16.113.2.75 `#define SUPVS_ADDRESS ((uint8_t) 0x1BU << 1U)`

Supply voltage status register address.

Definition at line 111 of file sbc\_uja113x.h.

16.113.2.76 `#define SYSIE_ADDRESS ((uint8_t) 0x04U << 1U)`

System interrupt enable register address.

Definition at line 87 of file sbc\_uja113x.h.

16.113.2.77 `#define SYSIS_ADDRESS ((uint8_t) 0x61U << 1U)`

System interrupt status register address.

Definition at line 182 of file sbc\_uja113x.h.

16.113.2.78 `#define TIE_ADDRESS ((uint8_t) 0x23U << 1U)`

Transceiver interrupt enable register address.

Definition at line 122 of file sbc\_uja113x.h.

16.113.2.79 `#define TIM1C_ADDRESS ((uint8_t) 0x50U << 1U)`

Timer 1 control register address.

Definition at line 167 of file sbc\_uja113x.h.

16.113.2.80 `#define TIM1DCC_ADDRESS ((uint8_t) 0x51U << 1U)`

Timer 1 duty cycle control register address.

Definition at line 168 of file sbc\_uja113x.h.

16.113.2.81 `#define TIM2C_ADDRESS ((uint8_t) 0x52U << 1U)`

Timer 2 control register address.

Definition at line 169 of file sbc\_uja113x.h.

16.113.2.82 `#define TIM2DCC_ADDRESS ((uint8_t) 0x53U << 1U)`

Timer 2 duty cycle control register address.

Definition at line 170 of file sbc\_uja113x.h.

16.113.2.83 `#define TIM3C_ADDRESS ((uint8_t) 0x54U << 1U)`

Timer 3 control register address.

Definition at line 171 of file sbc\_uja113x.h.

16.113.2.84 `#define TIM3DCC_ADDRESS ((uint8_t) 0x55U << 1U)`

Timer 3 duty cycle control register address.

Definition at line 172 of file sbc\_uja113x.h.

**16.113.2.85** `#define TIM4C_ADDRESS ((uint8_t) 0x56U << 1U)`

Timer 4 control register address.

Definition at line 173 of file sbc\_uja113x.h.

**16.113.2.86** `#define TIM4DCC_ADDRESS ((uint8_t) 0x57U << 1U)`

Timer 4 duty cycle control register address.

Definition at line 174 of file sbc\_uja113x.h.

**16.113.2.87** `#define TIS_ADDRESS ((uint8_t) 0x63U << 1U)`

Transceiver interrupt status register address.

Definition at line 184 of file sbc\_uja113x.h.

**16.113.2.88** `#define TS_ADDRESS ((uint8_t) 0x22U << 1U)`

Transceiver status register address.

Definition at line 121 of file sbc\_uja113x.h.

**16.113.2.89** `#define VBADC1_ADDRESS ((uint8_t) 0x15U << 1U)`

ADC conversion results for Vbat register 1 address.

Definition at line 105 of file sbc\_uja113x.h.

**16.113.2.90** `#define VBADC2_ADDRESS ((uint8_t) 0x16U << 1U)`

ADC conversion results for Vbat register 2 address.

Definition at line 106 of file sbc\_uja113x.h.

**16.113.2.91** `#define WTDC_ADDRESS ((uint8_t) 0x00U << 1U)`

Watchdog control register address.

Definition at line 83 of file sbc\_uja113x.h.

**16.113.2.92** `#define WTDS_ADDRESS ((uint8_t) 0x05U << 1U)`

Watchdog status register address.

Definition at line 88 of file sbc\_uja113x.h.

## 16.114 S32K142 SoC Header file

### 16.114.1 Detailed Description

This module covers the S32K142 SoC Header file.

#### Modules

- [Backward Compatibility Symbols for S32K142](#)  
*This module covers backward compatibility symbols.*
- [Interrupt vector numbers for S32K142](#)  
*This module covers interrupt number allocation.*
- [Peripheral access layer for S32K142](#)  
*This module covers all memory mapped register available on SoC.*

## 16.115 S32K142 System Files

This module covers the SoC support file for S32K142.

SystemInit method is called automatically from start-up code to do the minimum setup of the SoC. It disables the watchdog, enables FPU and the power mode protection if the corresponding feature macro is enabled.

SystemCoreClockUpdate method can be used at any time to update SystemCoreClock. It evaluates the clock register settings and calculates the current core clock.

SystemSoftwareReset method initiates a system reset.

## 16.116 SBC configuration control options (0x74)

### 16.116.1 Detailed Description

This section lists all the possible options of 0x74 register related to:

1. V1 reset threshold at start-up configuration
2. FNMC mode configuration
3. SDMC mode configuration
4. VEXT pin mode selection
5. Sleep mode support configuraion

See also

[UJA113X\\_SetMtpnv\(\)](#)

#### V1 reset threshold (defined by V1RTC) at start-up options (0x74:V1RTSUC[5:4])

- #define [V1RTSUC\\_90\\_NOM](#) 0x00U  
*V1 undervoltage detection at 90% of nominal value at start-up (V1RTC = 00)*
- #define [V1RTSUC\\_80\\_NOM](#) 0x10U  
*V1 undervoltage detection at 80% of nominal value at start-up (V1RTC = 01)*
- #define [V1RTSUC\\_70\\_NOM](#) 0x20U  
*V1 undervoltage detection at 70% of nominal value at start-up (V1RTC = 10)*
- #define [V1RTSUC\\_60\\_NOM](#) 0x30U  
*V1 undervoltage detection at 60% of nominal value at start-up (V1RTC = 11)*

#### Forced normal mode options (0x74:FNMC[3])

- #define [FNMC\\_DISABLE](#) 0x00U  
*Disable Forced Normal mode.*
- #define [FNMC\\_ENABLE](#) 0x08U  
*Enabled Forced Normal mode.*

#### Software Development mode options (0x74:SDMC[2])

- #define [SDMC\\_DISABLE](#) 0x00U  
*Disable Software Development mode.*
- #define [SDMC\\_ENABLE](#) 0x04U  
*Enable Software Development mode.*

#### V2 regulator configuration options (0x74:VEXTAC[1])

- #define [VEXTAC\\_SENSOR\\_SUPPLY](#) 0x00U  
*Regulator V2 can be used as a sensor supply via pin VEXT, provided pin V2 is left floating.*
- #define [VEXTAC\\_NO\\_PROTECTION](#) 0x02U  
*Regulator V2 is not protected against shorts to higher voltages; pin V2 must be shorted to pin VEXT.*

## Sleep mode configuration options (0x74:SLPC[0])

- `#define SLPC_SLEEP_SUPPORTED 0x00U`  
*Sleep mode is supported.*
- `#define SLPC_SLEEP_NOT_SUPPORTED 0x01U`  
*Sleep mode is not supported.*

## 16.116.2 Macro Definition Documentation

### 16.116.2.1 `#define FNMC_DISABLE 0x00U`

Disable Forced Normal mode.

Definition at line 1341 of file `sbc_uja113x.h`.

### 16.116.2.2 `#define FNMC_ENABLE 0x08U`

Enabled Forced Normal mode.

Definition at line 1342 of file `sbc_uja113x.h`.

### 16.116.2.3 `#define SDMC_DISABLE 0x00U`

Disable Software Development mode.

Definition at line 1348 of file `sbc_uja113x.h`.

### 16.116.2.4 `#define SDMC_ENABLE 0x04U`

Enable Software Development mode.

Definition at line 1349 of file `sbc_uja113x.h`.

### 16.116.2.5 `#define SLPC_SLEEP_NOT_SUPPORTED 0x01U`

Sleep mode is not supported.

Definition at line 1363 of file `sbc_uja113x.h`.

### 16.116.2.6 `#define SLPC_SLEEP_SUPPORTED 0x00U`

Sleep mode is supported.

Definition at line 1362 of file `sbc_uja113x.h`.

### 16.116.2.7 `#define V1RTSUC_60_NOM 0x30U`

V1 undervoltage detection at 60% of nominal value at start-up (V1RTC = 11)

Definition at line 1335 of file `sbc_uja113x.h`.

### 16.116.2.8 `#define V1RTSUC_70_NOM 0x20U`

V1 undervoltage detection at 70% of nominal value at start-up (V1RTC = 10)

Definition at line 1334 of file `sbc_uja113x.h`.

### 16.116.2.9 `#define V1RTSUC_80_NOM 0x10U`

V1 undervoltage detection at 80% of nominal value at start-up (V1RTC = 01)

Definition at line 1333 of file `sbc_uja113x.h`.



**16.116.2.10 #define V1RTSUC\_90\_NOM 0x00U**

V1 undervoltage detection at 90% of nominal value at start-up (V1RTC = 00)

Definition at line 1332 of file sbc\_uja113x.h.

**16.116.2.11 #define VEXTAC\_NO\_PROTECTION 0x02U**

Regulator V2 is not protected against shorts to higher voltages; pin V2 must be shorted to pin VEXT.

Definition at line 1356 of file sbc\_uja113x.h.

**16.116.2.12 #define VEXTAC\_SENSOR\_SUPPLY 0x00U**

Regulator V2 can be used as a sensor supply via pin VEXT, provided pin V2 is left floating.

Definition at line 1355 of file sbc\_uja113x.h.

## 16.117 Schedule management

### 16.117.1 Detailed Description

This group contains APIs that help users manage schedule tables in master node only.

#### Functions

- `I_u8 I_sch_tick (I_ifc_handle iii)`

*This function follows a schedule. When a frame becomes due, its transmission is initiated. When the end of the current schedule is reached, this function starts again at the beginning of the schedule.*

- `void I_sch_set (I_ifc_handle iii, I_schedule_handle schedule_iii, I_u8 entry)`

*Set up the next schedule to be followed by the I\_sch\_tick function for a certain interface. The new schedule will be activated as soon as the current schedule reaches its next schedule entry point.*

### 16.117.2 Function Documentation

#### 16.117.2.1 void I\_sch\_set ( I\_ifc\_handle iii, I\_schedule\_handle schedule\_iii, I\_u8 entry )

Set up the next schedule to be followed by the I\_sch\_tick function for a certain interface. The new schedule will be activated as soon as the current schedule reaches its next schedule entry point.

##### Parameters

in	iii	Interface name
in	schedule_iii	Schedule table for interface
in	entry	Entry to be set

##### Returns

void

Definition at line 73 of file lin\_common\_api.c.

#### 16.117.2.2 I\_u8 I\_sch\_tick ( I\_ifc\_handle iii )

This function follows a schedule. When a frame becomes due, its transmission is initiated. When the end of the current schedule is reached, this function starts again at the beginning of the schedule.

##### Parameters

in	Interface	name
----	-----------	------

##### Returns

Operation status

- Zero: if the next call of I\_sch\_tick will not start transmission of a frame.
- Non-Zero: if the next call of I\_sch\_tick will start transmission of a frame. The return value will in this case be the next schedule table entry's number (counted from the beginning of the schedule table) in the schedule table. The return value will be in range 1 to N if the schedule table has N entries.

Definition at line 240 of file lin\_common\_api.c.

## 16.118 Security PAL

### 16.118.1 Detailed Description

Security Peripheral Abstraction Layer.

#### Data Structures

- struct [security\\_user\\_config\\_t](#)

Define user configuration Implements : [security\\_user\\_config\\_t](#) Class. [More...](#)

#### Enumerations

- enum [security\\_instance\\_t](#) { SECURITY\_INSTANCE0 = 0U }

Define instances for SECURITY PAL Implements : [security\\_instance\\_t](#) Class.

- enum [security\\_key\\_id\\_t](#) {  
SECURITY\_SECRET\_KEY = 0x0U, SECURITY\_MASTER\_ECU = 0x1U, SECURITY\_BOOT\_MAC\_KEY = 0x2U, SECURITY\_BOOT\_MAC = 0x3U,  
SECURITY\_KEY\_1, SECURITY\_KEY\_2, SECURITY\_KEY\_3, SECURITY\_KEY\_4,  
SECURITY\_KEY\_5, SECURITY\_KEY\_6, SECURITY\_KEY\_7, SECURITY\_KEY\_8,  
SECURITY\_KEY\_9, SECURITY\_KEY\_10, SECURITY\_RAM\_KEY = 0xFU, SECURITY\_KEY\_11 = 0x14U,  
SECURITY\_KEY\_12, SECURITY\_KEY\_13, SECURITY\_KEY\_14, SECURITY\_KEY\_15,  
SECURITY\_KEY\_16, SECURITY\_KEY\_17 }

Defines the security keys Implements : [security\\_key\\_id\\_t](#) Class.

- enum [security\\_boot\\_flavor\\_t](#) { SECURITY\_BOOT\_STRICT = 0U, SECURITY\_BOOT\_SERIAL = 1U, SECURITY\_BOOT\_PARALLEL = 2U, SECURITY\_BOOT\_NOT\_DEFINED = 3U }

Defines the security boot flavor Implements : [security\\_boot\\_flavor\\_t](#) Class.

- enum [security\\_cmd\\_t](#) {  
SECURITY\_CMD\_ENC\_ECB = 1U, SECURITY\_CMD\_ENC\_CBC, SECURITY\_CMD\_DEC\_ECB, SECURITY\_CMD\_DEC\_CBC,  
SECURITY\_CMD\_GENERATE\_MAC, SECURITY\_CMD\_VERIFY\_MAC, SECURITY\_CMD\_LOAD\_KEY,  
SECURITY\_CMD\_LOAD\_PLAIN\_KEY,  
SECURITY\_CMD\_EXPORT\_RAM\_KEY, SECURITY\_CMD\_INIT\_RNG, SECURITY\_CMD\_EXTEND\_SEED,  
SECURITY\_CMD\_RND,  
SECURITY\_CMD\_RESERVED\_1, SECURITY\_CMD\_BOOT\_FAILURE, SECURITY\_CMD\_BOOT\_OK, SECURITY\_CMD\_GET\_ID,  
SECURITY\_CMD\_BOOT\_DEFINE, SECURITY\_CMD\_DBG\_CHAL, SECURITY\_CMD\_DBG\_AUTH, SECURITY\_CMD\_RESERVED\_2,  
SECURITY\_CMD\_RESERVED\_3, SECURITY\_CMD\_MP\_COMPRESS }

Defines the security command Implements : [security\\_cmd\\_t](#) Class.

#### Functions

- void [SECURITY\\_GetDefaultConfig](#) ([security\\_user\\_config\\_t](#) \*config)  
Initializes the configuration structure.
- status\_t [SECURITY\\_Init](#) ([security\\_instance\\_t](#) instance, const [security\\_user\\_config\\_t](#) \*config)  
Initializes the SECURITY module.
- status\_t [SECURITY\\_Deinit](#) ([security\\_instance\\_t](#) instance)  
De-initializes the SECURITY module.
- status\_t [SECURITY\\_EncryptEcbBlocking](#) ([security\\_instance\\_t](#) instance, [security\\_key\\_id\\_t](#) keyId, const uint8\_t \*plainText, uint32\_t msgLen, uint8\_t \*cipherText, uint32\_t timeout)  
ECB Encryption.
- status\_t [SECURITY\\_DecryptEcbBlocking](#) ([security\\_instance\\_t](#) instance, [security\\_key\\_id\\_t](#) keyId, const uint8\_t \*cipherText, uint32\_t msgLen, uint8\_t \*plainText, uint32\_t timeout)

*ECB Decryption.*

- status\_t [SECURITY\\_EncryptCbcBlocking](#) (security\_instance\_t instance, security\_key\_id\_t keyId, const uint8\_t \*plainText, uint32\_t msgLen, const uint8\_t \*iv, uint8\_t \*cipherText, uint32\_t timeout)

*CBC Decryption.*

- status\_t [SECURITY\\_DecryptCbcBlocking](#) (security\_instance\_t instance, security\_key\_id\_t keyId, const uint8\_t \*cipherText, uint32\_t msgLen, const uint8\_t \*iv, uint8\_t \*plainText, uint32\_t timeout)

*CBC Decryption.*

- status\_t [SECURITY\\_GenerateMacBlocking](#) (security\_instance\_t instance, security\_key\_id\_t keyId, const uint8\_t \*msg, uint64\_t msgLen, uint8\_t \*cmac, uint32\_t timeout)

*MAC Generation.*

- status\_t [SECURITY\\_VerifyMacBlocking](#) (security\_instance\_t instance, security\_key\_id\_t keyId, const uint8\_t \*msg, uint64\_t msgLen, const uint8\_t \*mac, uint16\_t macLen, bool \*verifStatus, uint32\_t timeout)

*MAC Verification.*

- status\_t [SECURITY\\_LoadKey](#) (security\_instance\_t instance, security\_key\_id\_t keyId, const uint8\_t \*m1, const uint8\_t \*m2, const uint8\_t \*m3, uint8\_t \*m4, uint8\_t \*m5, uint32\_t timeout)

*Load Key.*

- status\_t [SECURITY\\_LoadPlainKey](#) (security\_instance\_t instance, const uint8\_t \*plainKey, uint32\_t timeout)

*Load Plain Key.*

- status\_t [SECURITY\\_ExportRamKey](#) (security\_instance\_t instance, uint8\_t \*m1, uint8\_t \*m2, uint8\_t \*m3, uint8\_t \*m4, uint8\_t \*m5, uint32\_t timeout)

*Export RAM key.*

- status\_t [SECURITY\\_ExtendSeed](#) (security\_instance\_t instance, const uint8\_t \*entropy, uint32\_t timeout)

*Initialize Random Number Generator.*

- status\_t [SECURITY\\_InitRng](#) (security\_instance\_t instance, uint32\_t timeout)

*Initialize Random Number Generator.*

- status\_t [SECURITY\\_GenerateRnd](#) (security\_instance\_t instance, uint8\_t \*rnd, uint32\_t timeout)

*Generate RND.*

- status\_t [SECURITY\\_GetId](#) (security\_instance\_t instance, const uint8\_t \*challenge, uint8\_t \*uid, uint8\_t \*sreg, uint8\_t \*mac, uint32\_t timeout)

*Get ID.*

- status\_t [SECURITY\\_SecureBoot](#) (security\_instance\_t instance, uint32\_t bootImageSize, const uint8\_t \*bootImagePtr, uint32\_t timeout)

*Secure boot.*

- status\_t [SECURITY\\_BootFailure](#) (security\_instance\_t instance, uint32\_t timeout)

*Boot Failure.*

- status\_t [SECURITY\\_BootOk](#) (security\_instance\_t instance, uint32\_t timeout)

*Boot Ok.*

- status\_t [SECURITY\\_BootDefine](#) (security\_instance\_t instance, uint32\_t bootSize, security\_boot\_flavor\_t bootFlavor, uint32\_t timeout)

*Boot Define.*

- status\_t [SECURITY\\_DbgChal](#) (security\_instance\_t instance, uint8\_t \*challenge, uint32\_t timeout)

*Debug Challenge.*

- status\_t [SECURITY\\_DbgAuth](#) (security\_instance\_t instance, const uint8\_t \*authorization, uint32\_t timeout)

*Debug Authentication.*

- status\_t [SECURITY\\_MPCompress](#) (security\_instance\_t instance, const uint8\_t \*msg, uint32\_t msgLen, uint8\_t \*mpCompress, uint32\_t timeout)

*Miyaguchi-Prenell Compression.*

- status\_t [SECURITY\\_GenerateTrnd](#) (security\_instance\_t instance, uint8\_t \*trnd, uint32\_t timeout)

*Generate True Random Number.*

- status\_t [SECURITY\\_CancelCommand](#) (security\_instance\_t instance)

*Cancel Command.*

- status\_t [SECURITY\\_GetAsyncCmdStatus](#) (security\_instance\_t instance)

*Get asynchronous command status.*

- status\_t SECURITY\_EncryptEcb (security\_instance\_t instance, security\_key\_id\_t keyId, const uint8\_t \*plainText, uint32\_t msgLen, uint8\_t \*cipherText)

*Encrypt ECB.*

- status\_t SECURITY\_DecryptEcb (security\_instance\_t instance, security\_key\_id\_t keyId, const uint8\_t \*cipherText, uint32\_t msgLen, uint8\_t \*plainText)

*Decrypt ECB.*

- status\_t SECURITY\_EncryptCbc (security\_instance\_t instance, security\_key\_id\_t keyId, const uint8\_t \*plainText, uint32\_t msgLen, const uint8\_t \*iv, uint8\_t \*cipherText)

*Encrypt CBC.*

- status\_t SECURITY\_DecryptCbc (security\_instance\_t instance, security\_key\_id\_t keyId, const uint8\_t \*cipherText, uint32\_t msgLen, const uint8\_t \*iv, uint8\_t \*plainText)

*Decrypt CBC.*

- status\_t SECURITY\_GenerateMac (security\_instance\_t instance, security\_key\_id\_t keyId, const uint8\_t \*msg, uint64\_t msgLen, uint8\_t \*cmac)

*Generate MAC.*

- status\_t SECURITY\_VerifyMac (security\_instance\_t instance, security\_key\_id\_t keyId, const uint8\_t \*msg, uint64\_t msgLen, const uint8\_t \*mac, uint16\_t macLen, bool \*verifStatus)

*Verify MAC.*

## 16.118.2 Data Structure Documentation

### 16.118.2.1 struct security\_user\_config\_t

Define user configuration Implements : security\_user\_config\_t\_Class.

Definition at line 155 of file security\_pal.h.

#### Data Fields

- security\_callback\_t callback
- void \* callbackParam

#### Field Documentation

##### 16.118.2.1.1 security\_callback\_t callback

The callback invoked when an asynchronous command is completed

Definition at line 157 of file security\_pal.h.

##### 16.118.2.1.2 void\* callbackParam

User parameter for the command completion callback

Definition at line 158 of file security\_pal.h.

## 16.118.3 Enumeration Type Documentation

### 16.118.3.1 enum security\_boot\_flavor\_t

Defines the security boot flavor Implements : security\_boot\_flavor\_t\_Class.

#### Enumerator

**SECURITY\_BOOT\_STRICT**

**SECURITY\_BOOT\_SERIAL**

***SECURITY\_BOOT\_PARALLEL***  
***SECURITY\_BOOT\_NOT\_DEFINED***

Definition at line 97 of file security\_pal.h.

16.118.3.2 **enum security\_cmd\_t**

Defines the security command Implements : security\_cmd\_t\_Class.

Enumerator

***SECURITY\_CMD\_ENC\_ECB***  
***SECURITY\_CMD\_ENC\_CBC***  
***SECURITY\_CMD\_DEC\_ECB***  
***SECURITY\_CMD\_DEC\_CBC***  
***SECURITY\_CMD\_GENERATE\_MAC***  
***SECURITY\_CMD\_VERIFY\_MAC***  
***SECURITY\_CMD\_LOAD\_KEY***  
***SECURITY\_CMD\_LOAD\_PLAIN\_KEY***  
***SECURITY\_CMD\_EXPORT\_RAM\_KEY***  
***SECURITY\_CMD\_INIT\_RNG***  
***SECURITY\_CMD\_EXTEND\_SEED***  
***SECURITY\_CMD\_RND***  
***SECURITY\_CMD\_RESERVED\_1***  
***SECURITY\_CMD\_BOOT\_FAILURE***  
***SECURITY\_CMD\_BOOT\_OK***  
***SECURITY\_CMD\_GET\_ID***  
***SECURITY\_CMD\_BOOT\_DEFINE***  
***SECURITY\_CMD\_DBG\_CHAL***  
***SECURITY\_CMD\_DBG\_AUTH***  
***SECURITY\_CMD\_RESERVED\_2***  
***SECURITY\_CMD\_RESERVED\_3***  
***SECURITY\_CMD\_MP\_COMPRESS***

Definition at line 109 of file security\_pal.h.

16.118.3.3 **enum security\_instance\_t**

Define instances for SECURITY PAL Implements : security\_instance\_t\_Class.

Enumerator

***SECURITY\_INSTANCE0***

Definition at line 49 of file security\_pal.h.

16.118.3.4 **enum security\_key\_id\_t**

Defines the security keys Implements : security\_key\_id\_t\_Class.

Enumerator

***SECURITY\_SECRET\_KEY***

**SECURITY\_MASTER\_ECU**  
**SECURITY\_BOOT\_MAC\_KEY**  
**SECURITY\_BOOT\_MAC**  
**SECURITY\_KEY\_1**  
**SECURITY\_KEY\_2**  
**SECURITY\_KEY\_3**  
**SECURITY\_KEY\_4**  
**SECURITY\_KEY\_5**  
**SECURITY\_KEY\_6**  
**SECURITY\_KEY\_7**  
**SECURITY\_KEY\_8**  
**SECURITY\_KEY\_9**  
**SECURITY\_KEY\_10**  
**SECURITY\_RAM\_KEY**  
**SECURITY\_KEY\_11**  
**SECURITY\_KEY\_12**  
**SECURITY\_KEY\_13**  
**SECURITY\_KEY\_14**  
**SECURITY\_KEY\_15**  
**SECURITY\_KEY\_16**  
**SECURITY\_KEY\_17**

Definition at line 58 of file security\_pal.h.

#### 16.118.4 Function Documentation

16.118.4.1 **status\_t SECURITY\_BootDefine ( security\_instance\_t instance, uint32\_t bootSize, security\_boot\_flavor\_t bootFlavor, uint32\_t timeout )**

Boot Define.

Implements an extension of the SHE standard to define both the user boot size and boot method.

Parameters

in	<i>instance</i>	security module instance
in	<i>bootSize</i>	Number of blocks of 128-bit data to check on boot. Maximum size is 512k↔ Bytes.
in	<i>bootFlavor</i>	The boot method.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS↔ S_TIMEOUT is returned.

Returns

Error Code after command execution. Unsupported code if function is not available.

Definition at line 695 of file security\_pal.c.

16.118.4.2 **status\_t SECURITY\_BootFailure ( security\_instance\_t instance, uint32\_t timeout )**

Boot Failure.

Signals a failure detected during later stages of the boot process.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_TIMEOUT is returned.

**Returns**

Error Code after command execution.

Definition at line 641 of file security\_pal.c.

**16.118.4.3** `status_t SECURITY_BootOk ( security_instance_t instance, uint32_t timeout )`

Boot Ok.

Marks a successful boot verification during later stages of the boot process.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_TIMEOUT is returned.

**Returns**

Error Code after command execution.

Definition at line 668 of file security\_pal.c.

**16.118.4.4** `status_t SECURITY_CancelCommand ( security_instance_t instance )`

Cancel Command.

Cancels a previously initiated command.

**Parameters**

in	<i>instance</i>	security module instance
----	-----------------	--------------------------

**Returns**

STATUS\_SUCCESS

Definition at line 844 of file security\_pal.c.

**16.118.4.5** `status_t SECURITY_DbgAuth ( security_instance_t instance, const uint8_t * authorization, uint32_t timeout )`

Debug Authentication.

Erases all keys (actual and outdated) stored in NVM Memory if the authorization is confirmed.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>authorization</i>	Pointer to the 128-bit buffer containing the authorization value.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_TIMEOUT is returned.

**Returns**

Error Code after command execution.

Definition at line 756 of file security\_pal.c.



16.118.4.6 `status_t SECURITY_DbgChal ( security_instance_t instance, uint8_t * challenge, uint32_t timeout )`

Debug Challenge.

Obtains a random number which the user shall use along with the MASTER\_ECU\_KEY and UID to return an authorization request.

#### Parameters

in	<i>instance</i>	security module instance
out	<i>challenge</i>	Pointer to the 128-bit buffer where the challenge data will be stored.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_↔S_TIMEOUT is returned.

#### Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 728 of file security\_pal.c.

16.118.4.7 `status_t SECURITY_DecryptCbc ( security_instance_t instance, security_key_id_t keyId, const uint8_t * cipherText, uint32_t msgLen, const uint8_t * iv, uint8_t * plainText )`

Decrypt CBC.

Asynchronously performs the AES-128 decryption in CBC mode of the input cipher text buffer.

#### Parameters

in	<i>instance</i>	security module instance
in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>cipherText</i>	Pointer to the cipher text buffer.
in	<i>msgLen</i>	Number of bytes of cipher text message to be decrypted. It should be multiple of 16 bytes.
in	<i>iv</i>	Pointer to the initialization vector buffer.
out	<i>plainText</i>	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.

#### Returns

STATUS\_BUSY if another command is in execution, otherwise STATUS\_SUCCESS.

Definition at line 982 of file security\_pal.c.

16.118.4.8 `status_t SECURITY_DecryptCbcBlocking ( security_instance_t instance, security_key_id_t keyId, const uint8_t * cipherText, uint32_t msgLen, const uint8_t * iv, uint8_t * plainText, uint32_t timeout )`

CBC Decryption.

Perform AES-128 decryption in CBC mode of the input cipher text buffer.

#### Parameters

in	<i>instance</i>	security module instance
in	<i>keyId</i>	KeyID used to perform the cryptographic operation
in	<i>cipherText</i>	Pointer to the cipher text buffer.
in	<i>msgLen</i>	Number of bytes of plain text message to be encrypted. It is multiple of 16 bytes.

in	<i>iv</i>	Pointer to the initialization vector buffer.
out	<i>plainText</i>	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_↵S_TIMEOUT is returned.

#### Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 312 of file security\_pal.c.

**16.118.4.9** `status_t SECURITY_DecryptEcb ( security_instance_t instance, security_key_id_t keyId, const uint8_t * cipherText, uint32_t msgLen, uint8_t * plainText )`

Decrypt ECB.

Asynchronously performs the AES-128 decryption in ECB mode of the input cipher text buffer.

#### Parameters

in	<i>instance</i>	security module instance
in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>cipherText</i>	Pointer to the cipher text buffer.
in	<i>msgLen</i>	Number of bytes of cipher text message to be decrypted. It should be multiple of 16 bytes.
out	<i>plainText</i>	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.

#### Returns

STATUS\_BUSY if another command is in execution, otherwise STATUS\_SUCCESS.

Definition at line 923 of file security\_pal.c.

**16.118.4.10** `status_t SECURITY_DecryptEcbBlocking ( security_instance_t instance, security_key_id_t keyId, const uint8_t * cipherText, uint32_t msgLen, uint8_t * plainText, uint32_t timeout )`

ECB Decryption.

Perform AES-128 decryption in ECB mode of the input cipher text buffer.

#### Parameters

in	<i>instance</i>	security module instance
in	<i>keyId</i>	KeyID used to perform the cryptographic operation
in	<i>cipherText</i>	Pointer to the cipher text buffer.
in	<i>msgLen</i>	Number of bytes of plain text message to be encrypted. It is multiple of 16 bytes.
out	<i>plainText</i>	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_↵S_TIMEOUT is returned.

#### Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 251 of file security\_pal.c.

16.118.4.11 `status_t SECURITY_Deinit ( security_instance_t instance )`

De-initializes the SECURITY module.

This function de-initializes the requested SECURITY instance.

**Parameters**

in	<i>instance</i>	security module instance
----	-----------------	--------------------------

**Returns**

Error or success status returned by API

Definition at line 188 of file security\_pal.c.

**16.118.4.12** `status_t SECURITY_EncryptCbc ( security_instance_t instance, security_key_id_t keyId, const uint8_t * plainText, uint32_t msgLen, const uint8_t * iv, uint8_t * cipherText )`

Encrypt CBC.

Asynchronously performs the AES-128 encryption in CBC mode of the input plain text buffer.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>plainText</i>	Pointer to the plain text buffer.
in	<i>msgLen</i>	Number of bytes of plain text message to be encrypted. It should be multiple of 16 bytes.
in	<i>iv</i>	Pointer to the initialization vector buffer.
out	<i>cipherText</i>	Pointer to the cipher text buffer. The buffer shall have the same size as the plain text buffer.

**Returns**

STATUS\_BUSY if another command is in execution, otherwise STATUS\_SUCCESS.

Definition at line 952 of file security\_pal.c.

**16.118.4.13** `status_t SECURITY_EncryptCbcBlocking ( security_instance_t instance, security_key_id_t keyId, const uint8_t * plainText, uint32_t msgLen, const uint8_t * iv, uint8_t * cipherText, uint32_t timeout )`

CBC Decryption.

Perform AES-128 decryption in CBC mode of the input cipher text buffer.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>keyId</i>	KeyID used to perform the cryptographic operation
in	<i>plainText</i>	Pointer to the plain text buffer. The buffer shall have the same size as the cipher text buffer.
in	<i>msgLen</i>	Number of bytes of plain text message to be encrypted. It is multiple of 16 bytes.
in	<i>iv</i>	Pointer to the initialization vector buffer.
out	<i>cipherText</i>	Pointer to the cipher text buffer.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_S_TIMEOUT is returned.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 281 of file security\_pal.c.

16.118.4.14 `status_t SECURITY_EncryptEcb ( security_instance_t instance, security_key_id_t keyId, const uint8_t * plainText, uint32_t msgLen, uint8_t * cipherText )`

Encrypt ECB.

Asynchronously performs the AES-128 encryption in ECB mode of the input plain text buffer.

#### Parameters

in	<i>instance</i>	security module instance
in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>plainText</i>	Pointer to the plain text buffer.
in	<i>msgLen</i>	Number of bytes of plain text message to be encrypted. It should be multiple of 16 bytes.
out	<i>cipherText</i>	Pointer to the cipher text buffer. The buffer shall have the same size as the plain text buffer.

#### Returns

STATUS\_BUSY if another command is in execution, otherwise STATUS\_SUCCESS.

Definition at line 894 of file security\_pal.c.

16.118.4.15 `status_t SECURITY_EncryptEcbBlocking ( security_instance_t instance, security_key_id_t keyId, const uint8_t * plainText, uint32_t msgLen, uint8_t * cipherText, uint32_t timeout )`

ECB Encryption.

Perform AES-128 encryption in ECB mode of the input plain text buffer.

#### Parameters

in	<i>instance</i>	security module instance
in	<i>keyId</i>	KeyID used to perform the cryptographic operation
in	<i>plainText</i>	Pointer to the plain text buffer
in	<i>msgLen</i>	Number of bytes of plain text message to be encrypted. It is multiple of 16 bytes.
out	<i>cipherText</i>	Pointer to the cipher text buffer. The buffer shall have the same size as the plain text buffer.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_TIMEOUT is returned.

#### Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 221 of file security\_pal.c.

16.118.4.16 `status_t SECURITY_ExportRamKey ( security_instance_t instance, uint8_t * m1, uint8_t * m2, uint8_t * m3, uint8_t * m4, uint8_t * m5, uint32_t timeout )`

Export RAM key.

Exports the RAM\_KEY into a format protected by SECRET\_KEY.

#### Parameters

in	<i>instance</i>	security module instance
----	-----------------	--------------------------

out	<i>m1</i>	Pointer to a buffer where the M1 parameter will be exported.
out	<i>m2</i>	Pointer to a buffer where the M2 parameter will be exported.
out	<i>m3</i>	Pointer to a buffer where the M3 parameter will be exported.
out	<i>m4</i>	Pointer to a buffer where the M4 parameter will be exported.
out	<i>m5</i>	Pointer to a buffer where the M5 parameter will be exported.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_↵S_TIMEOUT is returned.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 467 of file security\_pal.c.

**16.118.4.17** `status_t SECURITY_ExtendSeed ( security_instance_t instance, const uint8_t * entropy, uint32_t timeout )`

Initialize Random Number Generator.

Extends the seed of the PRNG by compressing the former seed value and the supplied entropy into a new seed. This new seed is then to be used to generate a random number by invoking the CMD\_RND command. The random number generator must be initialized by CMD\_INIT\_RNG before the seed may be extended.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>entropy</i>	pointer to a 128-bit buffer containing the entropy.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_↵S_TIMEOUT is returned.

**Returns**

Error Code after command execution.

Definition at line 498 of file security\_pal.c.

**16.118.4.18** `status_t SECURITY_GenerateMac ( security_instance_t instance, security_key_id_t keyId, const uint8_t * msg, uint64_t msgLen, uint8_t * cmac )`

Generate MAC.

Asynchronously calculates the MAC of a given message using CMAC with AES-128.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>msg</i>	Pointer to the message buffer.
in	<i>msgLen</i>	Number of bits of message on which CMAC will be computed.
out	<i>cmac</i>	Pointer to the buffer containing the result of the CMAC computation.

**Returns**

STATUS\_BUSY if another command is in execution, otherwise STATUS\_SUCCESS.

Definition at line 1012 of file security\_pal.c.

**16.118.4.19** `status_t SECURITY_GenerateMacBlocking ( security_instance_t instance, security_key_id_t keyId, const uint8_t * msg, uint64_t msgLen, uint8_t * cmac, uint32_t timeout )`

MAC Generation.

Calculates MAC of a given message using CMAC with AES-128.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>msg</i>	Pointer to the message buffer.
in	<i>msgLen</i>	Number of bits of message on which CMAC will be computed.
out	<i>cmac</i>	Pointer to the buffer containing the result of the CMAC computation.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_↵S_TIMEOUT is returned.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_↵S_TIMEOUT is returned.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 342 of file security\_pal.c.

**16.118.4.20** `status_t SECURITY_GenerateRnd ( security_instance_t instance, uint8_t * rnd, uint32_t timeout )`

Generate RND.

Generates a vector of 128 random bits.

**Parameters**

in	<i>instance</i>	security module instance
out	<i>rnd</i>	Pointer to a 128-bit buffer where the generated random number has to be stored.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_↵S_TIMEOUT is returned.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 551 of file security\_pal.c.

**16.118.4.21** `status_t SECURITY_GenerateTrnd ( security_instance_t instance, uint8_t * trnd, uint32_t timeout )`

Generate True Random Number.

Generates a vector of 128 random bits using TRNG.

**Parameters**

in	<i>instance</i>	security module instance
out	<i>trnd</i>	Pointer to a 128-bit buffer where the generated random number is stored.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_↵S_TIMEOUT is returned.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.  
Unsupported code if function is not available.

Definition at line 816 of file security\_pal.c.

**16.118.4.22** `status_t SECURITY_GetAsyncCmdStatus ( security_instance_t instance )`

Get asynchronous command status.

Checks the status of the execution of an asynchronous command.

**Parameters**

in	<i>instance</i>	security module instance
----	-----------------	--------------------------

**Returns**

Error Code after command execution; STATUS\_BUSY if a command is still in progress.

Definition at line 869 of file security\_pal.c.

**16.118.4.23** void SECURITY\_GetDefaultConfig ( security\_user\_config\_t \* config )

Initializes the configuration structure.

This function initializes the configuration struct members to default values.

**Parameters**

out	<i>config</i>	configuration struct pointer
-----	---------------	------------------------------

Definition at line 126 of file security\_pal.c.

**16.118.4.24** status\_t SECURITY\_GetId ( security\_instance\_t instance, const uint8\_t \* challenge, uint8\_t \* uid, uint8\_t \* sreg, uint8\_t \* mac, uint32\_t timeout )

Get ID.

Returns the identity (UID) and the value of the status register protected by a MAC over a challenge and the data.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>challenge</i>	Pointer to the 128-bit buffer containing Challenge data.
out	<i>uid</i>	Pointer to 120 bit buffer where the UID will be stored.
out	<i>sreg</i>	Value of the status register.
out	<i>mac</i>	Pointer to the 128 bit buffer where the MAC generated over challenge and UID and status will be stored.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_S_TIMEOUT is returned.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 579 of file security\_pal.c.

**16.118.4.25** status\_t SECURITY\_Init ( security\_instance\_t instance, const security\_user\_config\_t \* config )

Initializes the SECURITY module.

This function initializes and enables the requested SECURITY instance.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>config</i>	pointer to security module configuration structure

**Returns**

Error or success status returned by API

Definition at line 141 of file security\_pal.c.



16.118.4.26 `status_t SECURITY_InitRng ( security_instance_t instance, uint32_t timeout )`

Initialize Random Number Generator.

Initializes the seed and derive a key for the PRNG.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_TIMEOUT is returned.

**Returns**

Error Code after command execution.

Definition at line 525 of file security\_pal.c.

**16.118.4.27** `status_t SECURITY_LoadKey ( security_instance_t instance, security_key_id_t keyId, const uint8_t * m1, const uint8_t * m2, const uint8_t * m3, uint8_t * m4, uint8_t * m5, uint32_t timeout )`

Load Key.

Updates an internal key per the SHE specification.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>keyId</i>	KeyID of the key to be updated.
in	<i>m1</i>	Pointer to the 128-bit M1 message containing the UID, Key ID and Authentication Key ID.
in	<i>m2</i>	Pointer to the 256-bit M2 message contains the new security flags, counter and the key value all encrypted using a derived key generated from the Authentication Key.
in	<i>m3</i>	Pointer to the 128-bit M3 message is a MAC generated over messages M1 and M2.
out	<i>m4</i>	Pointer to a 256 bits buffer where the computed M4 parameter is stored.
out	<i>m5</i>	Pointer to a 128 bits buffer where the computed M5 parameter is stored.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_TIMEOUT is returned.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 408 of file security\_pal.c.

**16.118.4.28** `status_t SECURITY_LoadPlainKey ( security_instance_t instance, const uint8_t * plainKey, uint32_t timeout )`

Load Plain Key.

Updates the RAM key memory slot with a 128-bit plaintext.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>plainKey</i>	Pointer to the 128-bit buffer containing the key that needs to be copied in RAM_KEY slot.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_TIMEOUT is returned.

**Returns**

Error Code after command execution.

Definition at line 440 of file security\_pal.c.

16.118.4.29 `status_t SECURITY_MPCompress ( security_instance_t instance, const uint8_t * msg, uint32_t msgLen, uint8_t * mpCompress, uint32_t timeout )`

Miyaguchi-Prenell Compression.

Compresses the given messages by accessing the Miyaguchi-Prenell compression feature with in the CSEc feature set.

#### Parameters

in	<i>instance</i>	security module instance
in	<i>msg</i>	Pointer to the messages to be compressed. Messages must be pre-processed per SHE specification if they do not already meet the full 128-bit block size requirement.
in	<i>msgLen</i>	The number of 128 bit messages to be compressed.
out	<i>mpCompress</i>	Pointer to the 128 bit buffer storing the compressed data.
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_↵S_TIMEOUT is returned.

#### Returns

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 785 of file security\_pal.c.

16.118.4.30 `status_t SECURITY_SecureBoot ( security_instance_t instance, uint32_t bootImageSize, const uint8_t * bootImagePtr, uint32_t timeout )`

Secure boot.

The function loads the command processor firmware and memory slot data and then executes the SHE secure boot protocol.

#### Parameters

in	<i>instance</i>	security module instance
in	<i>bootImageSize</i>	Boot image size (in bytes).
in	<i>bootImagePtr</i>	Boot image start address.

#### Note

Address passed in this parameter must be 32 bit aligned.

#### Parameters

in	<i>timeout</i>	Timeout in ms; the function returns STATUS_TIMEOUT if the command is not finished in the allocated period.
----	----------------	--

#### Returns

Error Code after command execution.

Definition at line 610 of file security\_pal.c.

16.118.4.31 `status_t SECURITY_VerifyMac ( security_instance_t instance, security_key_id_t keyId, const uint8_t * msg, uint64_t msgLen, const uint8_t * mac, uint16_t macLen, bool * verifStatus )`

Verify MAC.

Asynchronously verifies the MAC of a given message using CMAC with AES-128.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>msg</i>	Pointer to the message buffer.
in	<i>msgLen</i>	Number of bits of message on which CMAC will be computed.
in	<i>mac</i>	Pointer to the buffer containing the CMAC to be verified.
in	<i>macLen</i>	Number of bits of the CMAC to be compared. A macLength value of zero indicates that all 128-bits are compared.
out	<i>verifStatus</i>	Status of MAC verification command (true: verification operation passed, false: verification operation failed).

**Returns**

STATUS\_BUSY if another command is in execution, otherwise STATUS\_SUCCESS.

Definition at line 1044 of file security\_pal.c.

**16.118.4.32** `status_t SECURITY_VerifyMacBlocking ( security_instance_t instance, security_key_id_t keyId, const uint8_t * msg, uint64_t msgLen, const uint8_t * mac, uint16_t macLen, bool * verifStatus, uint32_t timeout )`

**MAC Verification.**

Verifies the MAC of a given message using CMAC with AES-128.

**Parameters**

in	<i>instance</i>	security module instance
in	<i>keyId</i>	KeyID used to perform the cryptographic operation.
in	<i>msg</i>	Pointer to the message buffer.
in	<i>msgLen</i>	Number of bits of message on which CMAC will be computed.
in	<i>mac</i>	Pointer to the buffer containing the CMAC to be verified.
in	<i>macLen</i>	Number of bits of the CMAC to be compared. A macLength value of zero indicates that all 128-bits are compared.
out	<i>verifStatus</i>	Status of MAC verification command (true: verification operation passed, false: verification operation failed).
in	<i>timeout</i>	Specifies the maximum time allowed for command completion, else STATUS_S_TIMEOUT is returned.

**Returns**

Error Code after command execution. Output parameters are valid if the error code is STATUS\_SUCCESS.

Definition at line 374 of file security\_pal.c.

## 16.119 Security Peripheral Abstraction Layer - SECURITY PAL

### 16.119.1 Detailed Description

The SECURITY PAL provides security features over specific modules like:

- > Cryptographic Services Engine (CSEc)
- > Hardware Security Module (HSM)

#### Features

- Secure cryptographic key storage
- AES-128 encryption and decryption
- AES-128 CMAC (Cipher-based Message Authentication Code) calculation and authentication
- ECB (Electronic Cypher Book) Mode - encryption and decryption
- CBC (Cipher Block Chaining) Mode - encryption and decryption
- True and Pseudo random number generation
- Miyaguchi-Prenell compression function
- Secure Boot Mode (user configurable)

#### How to use the SECURITY PAL in your application

The SECURITY PAL is designed to be used in conjunction with CSEc driver or HSM driver, based on hardware platform. The SECURITY PAL can't be used simultaneously over different driver types.

The following table contains the matching between platforms and available IPs:

IP/↔ MCU	S32↔ K116	S32↔ K118	S32↔ K142	S32↔ K144	S32↔ K142↔ W	S32↔ K144↔ W	S32↔ K146	S32↔ K148	S32↔ MTV	MP↔ C5746↔ C	MP↔ C5748↔ G
CS↔ EC	YES	YES	YES	YES	YES	YES	YES	YES	YES	NO	NO
HSM	NO	NO	NO	NO	NO	NO	NO	NO	NO	YES	YES

The SECURITY PAL includes file `security_pal_cfg.h`, which allows the user to specify which IP is used and how many resources are allocated (state structure). The following code example shows how to configure one instance for one available security module.

```
#ifndef SECURITY_PAL_CFG_H
#define SECURITY_PAL_CFG_H

/* Define which IP instance will be used in current project */
#define SECURITY_OVER_CSEC
#define NO_OF_CSEC_INSTS_FOR_SECURITY 1

#endif /* SECURITY_PAL_CFG_H */
```

In order to use the SECURITY modules, the initialization procedure must be completed. Using the [SECURITY\\_Init\(\)](#) function, the instance of the module is selected and configured using the user configuration structure.

The security features are available in two types: blocking and non-blocking. The blocking features have specified in their naming the 'blocking' attribute. All other functions are considered to be non-blocking. The blocking functions use the osif layer, providing timeout feature.

## Important Notes ##

- For advanced usage, user must verify the specific drivers documentation for CSEc or HSM.
- The SECURITY PAL enables the interrupts for the corresponding module. The application must configure the interrupts priorities.

- The SECURITY PAL API offers a "timeout" parameter for a number of functions. If "timeout" value is set to value '0', the returned status shall not be assumed as "STATUS\_TIMEOUT". This behaviour is given by the response time of the OS system tick and the execution time of the called function. Example: If OS system tick is set to 1ms, then the response time for a timeout set to '0', is between 0ms and 0.99ms. If the execution time of the function is 10us, then there is a high probability that the returned status shall be "STATUS\_SUCCESS".

### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\pal\src\security\security_pal.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc
${S32SDK_PATH}\platform\pal\inc
${S32SDK_PATH}\rtos\osif
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

#### Clock Manager Interrupt Manager (Interrupt) OS Interface (OSIF)

## Example code ##

```
static security_user_config_t g_SecurityUserConfig;

void SecurityCallback(uint32_t completedCmd, void *callbackParam)
{
    security_cmd_t securityCmd = (security_cmd_t)completedCmd;
    switch (securityCmd)
    {
        case SECURITY_CMD_ENC_ECB:
            /* Do something... */
            break;
        default:
            /* Error... */
            break;
    }
}

void main()
{
    static status_t status = STATUS_SUCCESS;
    static uint8_t rndBuf[16];

    g_SecurityUserConfig.callback = SecurityCallback;

    status = SECURITY_Init(SECURITY_INSTANCE0, &g_SecurityUserConfig);
    if(status != STATUS_SUCCESS)
    {
        /* Error... */
    }
    status = SECURITY_InitRng(SECURITY_INSTANCE0, TIMEOUT);
    if(status != STATUS_SUCCESS)
    {
        /* Error... */
    }
    status = SECURITY_GenerateRnd(SECURITY_INSTANCE0, rndBuf, TIMEOUT);
    if(status != STATUS_SUCCESS)
    {
        /* Error... */
    }

    while(1);
}
```

#### Modules

- [Security PAL](#)

*Security Peripheral Abstraction Layer.*

## 16.120 Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL)

### 16.120.1 Detailed Description

Serial Peripheral Interface - Peripheral Abstraction Layer.

The SPI PAL driver allows communication on an SPI bus. It was designed to be portable across all platforms and IPs which support SPI communication.

#### How to integrate DSPI in your application

Unlike the other drivers, SPI PAL modules need to include a configuration file named `spi_pal_cfg.h`, which allows the user to specify which IPs are used and how many resources are allocated for each of them (state structures). The following code example shows how to configure one instance for each available SPI IPs.

```
#ifndef SPI_PAL_cfg_H
#define SPI_PAL_cfg_H

/* Define which IP instance will be used in current project */
#define SPI_OVER_LPSP
#define SPI_OVER_FLEXIO
#define SPI_OVER_DSPI

/* Define the resources necessary for current project */
#define NO_OF_LPSP_INSTS_FOR_SPI 1U
#define NO_OF_FLEXIO_INSTS_FOR_SPI 1U
#define NO_OF_DSPI_INSTS_FOR_SPI 1U
#endif /* SPI_PAL_cfg_H */
```

The following table contains the matching between platforms and available IPs

IP/ M CU	S32 K118	S32 K116	S32 K142	S32 K144	S32 K146	S32 K148	S32 K142 W	S32 K144 W	M P C5748 G	M P C5746 C	M P C5744 P	S32 R274	S32 R372
FL E XIO	Y ES	Y ES	Y ES	Y ES	Y ES	Y ES	Y ES	Y ES	NO	NO	NO	NO	NO
LP S PI	Y ES	Y ES	Y ES	Y ES	Y ES	Y ES	Y ES	Y ES	NO	NO	NO	NO	NO
D S PI/ S PI	NO	NO	NO	NO	NO	NO	NO	NO	Y ES	Y ES	Y ES	Y ES	Y ES

In order to use the SPI driver it must be first initialized in either master or slave mode, using functions [SPI\\_MasterInit\(\)](#) or [SPI\\_SlaveInit\(\)](#). Once initialized, it cannot be initialized again for the same SPI module instance until it is de-initialized, using [SPI\\_SlaveDeinit\(\)](#) or [SPI\\_MasterDeinit](#). Different SPI module instances can work independently of each other.

In each mode (master/slave) are available two types of transfers: blocking and non-blocking. The functions which initiate blocking transfers will configure the time out for transmission. If time expires [SPI\\_MasterTransferBlocking\(\)](#) or [SPI\\_SlaveTransferBlocking\(\)](#) will return error and the transmission will be aborted.

The configuration structure includes a special field named extension. It will be used only for SPI transfers over FLEXIO and should contain a pointer to [extension\\_flexio\\_for\\_spi\\_t](#) structure. The purpose of this structure is to configure which FLEXIO pins are used by the applications and their functionality (MISO, MOSI, SCK, SS). One FLEXIO hardware instance can implements spi, so if instType is SPI\_INST\_TYPE\_FLEXIO instIdx can be 0 or 1.

If device name is from MPC574xP and S32Rx7x families it can't be used as master in DMA mode and PAL will automatically switch the functionality to interrupt mode. If DMA mode is mandatory please use DSPI driver.



### Important Notes

- The driver enables the interrupts for the corresponding module, but any interrupt priority setting must be done by the application.

### ## Example code ##

```

/* Configure SPI master */
spi_master_t spi0_MasterConfig0 =
{
    .baudRate      = 100000,
    .ssPolarity    = SPI_ACTIVE_HIGH,
    .frameSize     = 8,
    .clockPhase    = READ_ON_ODD_EDGE,
    .clockPolarity = SPI_ACTIVE_HIGH,
    .bitOrder      = SPI_TRANSFER_MSB_FIRST,
    .transferType  = SPI_USING_INTERRUPTS,
    .rxDMAChannel  = 255,
    .txDMAChannel  = 255,
    .callback      = NULL,
    .callbackParam = NULL,
    .ssPin         = 0,
    .extension     = NULL
};

/* Configure FLEXIO pins routing */
extension_flexio_for_spi_t extension;
extension.misoPin = 0;
extension.mosiPin = 1;
extension.sckPin = 2;
extension.ssPin = 3;
spi0_MasterConfig0.extension = &extension;

/* Configure instances used in this example */
spi_instance_t lpspiInstance, flexioInstance;
lpspiInstance.instIdx = 0U;
lpspiInstance.instType = SPI_INST_TYPE_LPSPI;
flexioInstance.instIdx = 1U;
lpspiInstance.instType = SPI_INST_TYPE_FLEXIO;

/* Buffers */
uint8_t tx[5] = {1,2,3,4,5};
uint8_t rx[5];

/* Initializes SPI master for LPSPI 0 and send 5 frames */
SPI_MasterInit(&lpspiInstance, &spi0_MasterConfig0);
SPI_MasterTransfer(&lpspiInstance, tx, rx, 5);
/* Initializes SPI master for FLEXIO 0 and send 5 frames */
SPI_MasterInit(&flexioInstance, &spi1_MasterConfig0);
SPI_MasterTransfer(&flexioInstance, tx, rx, 5);

```

### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
{S32SDK_PATH}\platform\pal\src\spi\spi_pal.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
{S32SDK_PATH}\platform\pal\inc
spi_pal_cfg.h path
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

Low Power Serial Peripheral Interface (LPSPI) flexio\_spi Clock Manager OS Interface (OSIF) Interrupt Manager (Interrupt) Enhanced Direct Memory Access (eDMA)

## Data Structures

- struct [spi\\_master\\_t](#)  
Defines the configuration structure for SPI master Implements : [spi\\_master\\_t\\_Class](#). [More...](#)
- struct [spi\\_slave\\_t](#)  
Defines the configuration structure for SPI slave Implements : [spi\\_slave\\_t\\_Class](#). [More...](#)
- struct [extension\\_flexio\\_for\\_spi\\_t](#)  
Defines the extension structure for the SPI over FLEXIO Implements : [extension\\_flexio\\_for\\_spi\\_t\\_Class](#). [More...](#)

## Enumerations

- enum [spi\\_transfer\\_type\\_t](#) { [SPI\\_USING\\_DMA](#) = 0U, [SPI\\_USING\\_INTERRUPTS](#) = 1U }  
Defines the mechanism to update the rx or tx buffers Implements : [spi\\_transfer\\_type\\_t\\_Class](#).
- enum [spi\\_polarity\\_t](#) { [SPI\\_ACTIVE\\_HIGH](#) = 0U, [SPI\\_ACTIVE\\_LOW](#) = 1U }  
Defines the polarity of signals Implements : [spi\\_polarity\\_t\\_Class](#).
- enum [spi\\_clock\\_phase\\_t](#) { [READ\\_ON\\_ODD\\_EDGE](#) = 0U, [READ\\_ON\\_EVEN\\_EDGE](#) = 1U }  
Defines the edges used for sampling and shifting Implements : [spi\\_clock\\_phase\\_t\\_Class](#).
- enum [spi\\_transfer\\_bit\\_order\\_t](#) { [SPI\\_TRANSFER\\_MSB\\_FIRST](#) = 0U, [SPI\\_TRANSFER\\_LSB\\_FIRST](#) = 1U }  
Defines the bit order Implements : [spi\\_transfer\\_bit\\_order\\_t\\_Class](#).

## Functions

- status\_t [SPI\\_MasterInit](#) (const [spi\\_instance\\_t](#) \*const instance, const [spi\\_master\\_t](#) \*config)  
Initializes the SPI module in master mode.
- status\_t [SPI\\_Slavelnit](#) (const [spi\\_instance\\_t](#) \*const instance, const [spi\\_slave\\_t](#) \*config)  
Initializes the SPI module in slave mode.
- status\_t [SPI\\_SetSS](#) (const [spi\\_instance\\_t](#) \*const instance, uint8\_t ss)  
Update the SS.
- status\_t [SPI\\_MasterTransfer](#) (const [spi\\_instance\\_t](#) \*const instance, const void \*txBuffer, void \*rxBuffer, uint16\_t numberOfFrames)  
Initializes a non-blocking master transfer.
- status\_t [SPI\\_MasterTransferBlocking](#) (const [spi\\_instance\\_t](#) \*const instance, const void \*txBuffer, void \*rxBuffer, uint16\_t numberOfFrames, uint16\_t timeout)  
Initializes a blocking master transfer.
- status\_t [SPI\\_SlaveTransfer](#) (const [spi\\_instance\\_t](#) \*const instance, const void \*txBuffer, void \*rxBuffer, uint16\_t numberOfFrames)  
Initializes a non-blocking slave transfer.
- status\_t [SPI\\_SlaveTransferBlocking](#) (const [spi\\_instance\\_t](#) \*const instance, const void \*txBuffer, void \*rxBuffer, uint16\_t numberOfFrames, uint16\_t timeout)  
Initializes a blocking slave transfer.
- status\_t [SPI\\_GetStatus](#) (const [spi\\_instance\\_t](#) \*const instance)  
Gets the status of the last transfer.
- status\_t [SPI\\_GetDefaultMasterConfig](#) ([spi\\_master\\_t](#) \*config)  
Gets the default configuration structure for master.
- status\_t [SPI\\_GetDefaultSlaveConfig](#) ([spi\\_slave\\_t](#) \*config)  
Gets the default configuration structure for slave.
- status\_t [SPI\\_MasterDeinit](#) (const [spi\\_instance\\_t](#) \*const instance)  
De-initializes the spi master module.
- status\_t [SPI\\_SlaveDeinit](#) (const [spi\\_instance\\_t](#) \*const instance)  
De-initializes the spi slave module.
- status\_t [SPI\\_MasterSetDelay](#) (const [spi\\_instance\\_t](#) \*const instance, uint32\_t delayBetweenTransfers, uint32\_t delaySCKtoPCS, uint32\_t delayPCStoSCK)  
Configures the SPI\_PAL master mode bus timing delay options.

## 16.120.2 Data Structure Documentation

### 16.120.2.1 struct spi\_master\_t

Defines the configuration structure for SPI master Implements : spi\_master\_t\_Class.

Definition at line 82 of file spi\_pal.h.

#### Data Fields

- uint32\_t [baudRate](#)
- uint8\_t [frameSize](#)
- [spi\\_transfer\\_bit\\_order\\_t](#) [bitOrder](#)
- [spi\\_polarity\\_t](#) [clockPolarity](#)
- [spi\\_polarity\\_t](#) [ssPolarity](#)
- [spi\\_clock\\_phase\\_t](#) [clockPhase](#)
- uint8\_t [ssPin](#)
- [spi\\_transfer\\_type\\_t](#) [transferType](#)
- uint8\_t [rxDMAChannel](#)
- uint8\_t [txDMAChannel](#)
- [spi\\_callback\\_t](#) [callback](#)
- void \* [callbackParam](#)
- void \* [extension](#)

#### Field Documentation

##### 16.120.2.1.1 uint32\_t baudRate

Clock frequency

Definition at line 84 of file spi\_pal.h.

##### 16.120.2.1.2 spi\_transfer\_bit\_order\_t bitOrder

Select if first bit is MSB or LSB

Definition at line 86 of file spi\_pal.h.

##### 16.120.2.1.3 spi\_callback\_t callback

Select the callback to transfer complete

Definition at line 94 of file spi\_pal.h.

##### 16.120.2.1.4 void\* callbackParam

Select additional callback parameters if it's necessary

Definition at line 95 of file spi\_pal.h.

##### 16.120.2.1.5 spi\_clock\_phase\_t clockPhase

Select clock edges for sampling and shifting

Definition at line 89 of file spi\_pal.h.

##### 16.120.2.1.6 spi\_polarity\_t clockPolarity

Select polarity for Clock

Definition at line 87 of file spi\_pal.h.

**16.120.2.1.7 void\* extension**

This field will be used to add extra settings to the basic configuration like FlexIO pins

Definition at line 96 of file spi\_pal.h.

**16.120.2.1.8 uint8\_t frameSize**

Size of frame in bits

Definition at line 85 of file spi\_pal.h.

**16.120.2.1.9 uint8\_t rxDMAChannel**

Channel number for DMA rx channel

Definition at line 92 of file spi\_pal.h.

**16.120.2.1.10 uint8\_t ssPin**

Select which SS is used

Definition at line 90 of file spi\_pal.h.

**16.120.2.1.11 spi\_polarity\_t ssPolarity**

Select polarity for SS

Definition at line 88 of file spi\_pal.h.

**16.120.2.1.12 spi\_transfer\_type\_t transferType**

Select if buffers are managed by internal interrupt handler or by DMA

Definition at line 91 of file spi\_pal.h.

**16.120.2.1.13 uint8\_t txDMAChannel**

Channel number for DMA tx channel

Definition at line 93 of file spi\_pal.h.

**16.120.2.2 struct spi\_slave\_t**

Defines the configuration structure for SPI slave Implements : spi\_slave\_t\_Class.

Definition at line 103 of file spi\_pal.h.

**Data Fields**

- [uint8\\_t frameSize](#)
- [spi\\_transfer\\_bit\\_order\\_t bitOrder](#)
- [spi\\_polarity\\_t clockPolarity](#)
- [spi\\_polarity\\_t ssPolarity](#)
- [spi\\_clock\\_phase\\_t clockPhase](#)
- [spi\\_transfer\\_type\\_t transferType](#)
- [uint8\\_t rxDMAChannel](#)
- [uint8\\_t txDMAChannel](#)
- [spi\\_callback\\_t callback](#)
- void \* [callbackParam](#)
- void \* [extension](#)

**Field Documentation**

**16.120.2.2.1 spi\_transfer\_bit\_order\_t bitOrder**

Select if first bit is MSB or LSB

Definition at line 106 of file spi\_pal.h.

**16.120.2.2.2 spi\_callback\_t callback**

Select the callback to transfer complete

Definition at line 113 of file spi\_pal.h.

**16.120.2.2.3 void\* callbackParam**

Select additional callback parameters if it's necessary

Definition at line 114 of file spi\_pal.h.

**16.120.2.2.4 spi\_clock\_phase\_t clockPhase**

Select clock edges for sampling and shifting

Definition at line 109 of file spi\_pal.h.

**16.120.2.2.5 spi\_polarity\_t clockPolarity**

Select polarity for Clock

Definition at line 107 of file spi\_pal.h.

**16.120.2.2.6 void\* extension**

This field will be used to add extra settings to the basic configuration like FlexIO

Definition at line 115 of file spi\_pal.h.

**16.120.2.2.7 uint8\_t frameSize**

Size of frame in bits

Definition at line 105 of file spi\_pal.h.

**16.120.2.2.8 uint8\_t rxDMAChannel**

Channel number for DMA rx channel

Definition at line 111 of file spi\_pal.h.

**16.120.2.2.9 spi\_polarity\_t ssPolarity**

Select polarity for SS

Definition at line 108 of file spi\_pal.h.

**16.120.2.2.10 spi\_transfer\_type\_t transferType**

Select if buffers are managed by internal interrupt handler or by DMA

Definition at line 110 of file spi\_pal.h.

**16.120.2.2.11 uint8\_t txDMAChannel**

Channel number for DMA tx channel

Definition at line 112 of file spi\_pal.h.

### 16.120.2.3 struct extension\_flexio\_for\_spi\_t

Defines the extension structure for the SPI over FLEXIO Implements : extension\_flexio\_for\_spi\_t\_Class.

Definition at line 123 of file spi\_pal.h.

#### Data Fields

- uint8\_t [mosiPin](#)
- uint8\_t [misoPin](#)
- uint8\_t [sckPin](#)
- uint8\_t [ssPin](#)

#### Field Documentation

##### 16.120.2.3.1 uint8\_t misoPin

FlexIO pin for MISO

Definition at line 126 of file spi\_pal.h.

##### 16.120.2.3.2 uint8\_t mosiPin

FlexIO pin for MOSI

Definition at line 125 of file spi\_pal.h.

##### 16.120.2.3.3 uint8\_t sckPin

FlexIO pin for SCK

Definition at line 127 of file spi\_pal.h.

##### 16.120.2.3.4 uint8\_t ssPin

FlexIO pin for SS

Definition at line 128 of file spi\_pal.h.

### 16.120.3 Enumeration Type Documentation

#### 16.120.3.1 enum spi\_clock\_phase\_t

Defines the edges used for sampling and shifting Implements : spi\_clock\_phase\_t\_Class.

#### Enumerator

**READ\_ON\_ODD\_EDGE** The SPI signal is read on odd edges of SCK and counting starts after SS activation

**READ\_ON\_EVEN\_EDGE** The SPI signal is read on even edges of SCK and counting starts after SS activation

Definition at line 61 of file spi\_pal.h.

#### 16.120.3.2 enum spi\_polarity\_t

Defines the polarity of signals Implements : spi\_polarity\_t\_Class.

#### Enumerator

**SPI\_ACTIVE\_HIGH** The signal is active high

**SPI\_ACTIVE\_LOW** The signal is active low

Definition at line 51 of file spi\_pal.h.

## 16.120.3.3 enum spi\_transfer\_bit\_order\_t

Defines the bit order Implements : spi\_transfer\_bit\_order\_t\_Class.

## Enumerator

**SPI\_TRANSFER\_MSB\_FIRST** Transmit data starting with most significant bit

**SPI\_TRANSFER\_LSB\_FIRST** Transmit data starting with least significant bit

Definition at line 71 of file spi\_pal.h.

## 16.120.3.4 enum spi\_transfer\_type\_t

Defines the mechanism to update the rx or tx buffers Implements : spi\_transfer\_type\_t\_Class.

## Enumerator

**SPI\_USING\_DMA** The driver will use DMA to perform SPI transfer

**SPI\_USING\_INTERRUPTS** The driver will use interrupts to perform SPI transfer

Definition at line 41 of file spi\_pal.h.

## 16.120.4 Function Documentation

## 16.120.4.1 status\_t SPI\_GetDefaultMasterConfig ( spi\_master\_t \* config )

Gets the default configuration structure for master.

The default configuration structure is:

## Parameters

out	config	Pointer to configuration structure
-----	--------	------------------------------------

## Returns

Error or success status returned by API

Definition at line 734 of file spi\_pal.c.

## 16.120.4.2 status\_t SPI\_GetDefaultSlaveConfig ( spi\_slave\_t \* config )

Gets the default configuration structure for slave.

The default configuration structure is:

## Parameters

out	config	Pointer to configuration structure
-----	--------	------------------------------------

## Returns

Error or success status returned by API

Definition at line 759 of file spi\_pal.c.

## 16.120.4.3 status\_t SPI\_GetStatus ( const spi\_instance\_t \* const instance )

Gets the status of the last transfer.

This function return the status of the last transfer. Using this function the user can check if the transfer is still in progress or if time-out event occurred.

**Parameters**

in	<i>instance</i>	The name of the instance
in	<i>txBuffer</i>	Pointer to tx buffer.
in	<i>rxBuffer</i>	Pointer to rx buffer.
in	<i>numberOfFrames</i>	Number of frames sent/received
in	<i>timeout</i>	Transfer time-out in ms

**Returns**

Error or success status returned by API

Definition at line 946 of file spi\_pal.c.

#### 16.120.4.4 `status_t SPI_MasterDeinit ( const spi_instance_t *const instance )`

De-initializes the spi master module.

This function de-initialized the spi master module.

**Parameters**

in	<i>instance</i>	The name of the instance
----	-----------------	--------------------------

**Returns**

Error or success status returned by API

Definition at line 782 of file spi\_pal.c.

#### 16.120.4.5 `status_t SPI_MasterInit ( const spi_instance_t *const instance, const spi_master_t * config )`

Initializes the SPI module in master mode.

This function initializes and enables the requested SPI module in master mode, configuring the bus parameters.

**Parameters**

in	<i>instance</i>	The name of the instance
in	<i>config</i>	The configuration structure

**Returns**

Error or success status returned by API

Definition at line 248 of file spi\_pal.c.

#### 16.120.4.6 `status_t SPI_MasterSetDelay ( const spi_instance_t *const instance, uint32_t delayBetweenTransfers, uint32_t delaySCKtoPCS, uint32_t delayPCStoSCK )`

Configures the SPI\_PAL master mode bus timing delay options.

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 SPI\_PAL module has been initialized for master mode. The timings are adjusted in terms of microseconds. The bus timing delays that can be adjusted are listed below:

SCK to PCS Delay: Adjustable delay option between the last edge of SCK to the de-assertion of the PCS signal.

PCS to SCK Delay: Adjustable delay option between the assertion of the PCS signal to the first SCK edge.

Delay between Transfers: 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.

Definition at line 1014 of file spi\_pal.c.



**16.120.4.7** `status_t SPI_MasterTransfer ( const spi_instance_t *const instance, const void * txBuffer, void * rxBuffer, uint16_t numberOfFrames )`

Initializes a non-blocking master transfer.

This function initializes a non-blocking master transfer.

#### Parameters

in	<i>instance</i>	The name of the instance
in	<i>txBuffer</i>	Pointer to tx buffer.
in	<i>rxBuffer</i>	Pointer to rx buffer.
in	<i>numberOfFrames</i>	Number of frames sent/received

#### Returns

Error or success status returned by API

Definition at line 373 of file spi\_pal.c.

**16.120.4.8** `status_t SPI_MasterTransferBlocking ( const spi_instance_t *const instance, const void * txBuffer, void * rxBuffer, uint16_t numberOfFrames, uint16_t timeout )`

Initializes a blocking master transfer.

This function initializes a blocking master transfer.

#### Parameters

in	<i>instance</i>	The name of the instance
in	<i>txBuffer</i>	Pointer to tx buffer.
in	<i>rxBuffer</i>	Pointer to rx buffer.
in	<i>numberOfFrames</i>	Number of frames sent/received
in	<i>timeout</i>	Transfer time-out in ms

#### Returns

Error or success status returned by API

Definition at line 436 of file spi\_pal.c.

**16.120.4.9** `status_t SPI_SetSS ( const spi_instance_t *const instance, uint8_t ss )`

Update the SS.

This function changes the SS, if this feature is available.

#### Parameters

in	<i>instance</i>	The name of the instance
in	<i>ss</i>	The number of SS

#### Returns

Error or success status returned by API

Definition at line 900 of file spi\_pal.c.

**16.120.4.10** `status_t SPI_SlaveDeinit ( const spi_instance_t *const instance )`

De-initializes the spi slave module.

This function de-initialized the spi slave module.

**Parameters**

<i>in</i>	<i>instance</i>	The name of the instance
-----------	-----------------	--------------------------

**Returns**

Error or success status returned by API

Definition at line 844 of file spi\_pal.c.

**16.120.4.11** `status_t SPI_SlaveInit ( const spi_instance_t *const instance, const spi_slave_t * config )`

Initializes the SPI module in slave mode.

This function initializes and enables the requested SPI module in slave mode, configuring the bus parameters.

**Parameters**

<i>in</i>	<i>instance</i>	The name of the instance
<i>in</i>	<i>config</i>	The configuration structure

**Returns**

Error or success status returned by API

Definition at line 499 of file spi\_pal.c.

**16.120.4.12** `status_t SPI_SlaveTransfer ( const spi_instance_t *const instance, const void * txBuffer, void * rxBuffer, uint16_t numberOfFrames )`

Initializes a non-blocking slave transfer.

This function initializes a non-blocking slave transfer.

**Parameters**

<i>in</i>	<i>instance</i>	The name of the instance
<i>in</i>	<i>txBuffer</i>	Pointer to tx buffer.
<i>in</i>	<i>rxBuffer</i>	Pointer to rx buffer.
<i>in</i>	<i>numberOfFrames</i>	Number of frames sent/received

**Returns**

Error or success status returned by API

Definition at line 608 of file spi\_pal.c.

**16.120.4.13** `status_t SPI_SlaveTransferBlocking ( const spi_instance_t *const instance, const void * txBuffer, void * rxBuffer, uint16_t numberOfFrames, uint16_t timeout )`

Initializes a blocking slave transfer.

This function initializes a blocking slave transfer.

**Parameters**

<i>in</i>	<i>instance</i>	The name of the instance
-----------	-----------------	--------------------------

in	<i>txBuffer</i>	Pointer to tx buffer.
in	<i>rxBuffer</i>	Pointer to rx buffer.
in	<i>numberOf↔ Frames</i>	Number of frames sent/received
in	<i>timeout</i>	Transfer time-out in ms

**Returns**

Error or success status returned by API

Definition at line 671 of file spi\_pal.c.

## 16.121 Signal interaction

This group contains APIs that help users interact with signals of LIN node.

## 16.122 SoC Header file (SoC Header )

### 16.122.1 Detailed Description

This module covers SoC Header file.

This section describes the functionality supported by the header file. For usage please see `soc_header_usage`

#### Modules

- [S32K142 SoC Header file](#)

*This module covers the S32K142 SoC Header file.*

## 16.123 SoC Support

### 16.123.1 Detailed Description

This module covers SoC support files.

This section describes the files that are used for supporting various SoCs.

The support files are:

1. Linker files
2. Start-up files
3. SVD file
4. Header files

#### Linker files

Linker files are used to control the linkage part of the project compilation and contain details regarding the following:

1. memory areas definition (type and ranges)
2. data and code segments definition and their mapping to the memory areas.

linker configuration files are provided for all supported linkers. Please see [Build Tools](#) for details.

#### Start-up files

Start-up files are used to control the SoC bring-up part and contain:

1. interrupt vector allocation
2. start-up code and routines

Start-up files are provided for all supported compilers. Please see [Build Tools](#) for details.

#### SVD file

SVD file contains details about registers and can be used with an IDE to allow mapping of memory location to the register definition and information.

#### Header file

For each SoC there are two header files provided in the SDK:

1. `<SoC_name>.h`
2. `<SoC_name>_features.h`

The `<SoC_name>.h` file contains information related to registers that is used by the SDK drivers and code. The `<SoC_name>_features.h` contains information related to the integration of modules in the SoC.

#### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\devices\<SoC_name>\startup\system_<SoC_name>.c  
${S32SDK_PATH}\platform\devices\startup.c  
${S32SDK_PATH}\platform\devices\<SoC_name>\startup\<Toolchain>\startup_<SoC_name>.S
```

### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\devices  
${S32SDK_PATH}\platform\devices\<SoC_name>\  
${S32SDK_PATH}\platform\devices\startup\  
${S32SDK_PATH}\platform\devices\<SoC_name>\include\  
${S32SDK_PATH}\platform\devices\common
```

### Compile symbols

```
CPU_S32K144HFT0VLLT for S32K144  
CPU_<SoC_name>
```

### Dependencies

No special dependencies are required for this component

### Limitations

IAR: Function alignment is not supported using `ALIGNED()` macro.

### Modules

- [S32K142 System Files](#)

*This module covers the SoC support file for S32K142.*

## 16.124 Start-up control configuration options (0x73)

### 16.124.1 Detailed Description

This section lists all the possible options of 0x73 register related to:

- Reset pulse width on RSTN pin
- V2 start-up configuration
- HVIO 4 - 2 pin mode selection

See also

[UJA113X\\_SetMtpnv\(\)](#)

#### RSTN output reset pulse width configuration options (0x73:RLC[5:4])

- #define [RLC\\_PULSE\\_WIDTH\\_20\\_25\\_MS](#) 0x00U  
*tw(rst) = 20 ms to 25 ms*
- #define [RLC\\_PULSE\\_WIDTH\\_10\\_12\\_MS](#) 0x10U  
*tw(rst) = 10 ms to 12.5 ms*
- #define [RLC\\_PULSE\\_WIDTH\\_03\\_05\\_MS](#) 0x20U  
*tw(rst) = 3.6 ms to 5 ms*
- #define [RLC\\_PULSE\\_WIDTH\\_01\\_MS](#) 0x30U  
*tw(rst) = 1 ms to 1.5 ms*

#### V2 start-up control configuration options (0x73:V2SUC[3])

See also

[uja113x\\_rc\\_v2c\\_t](#)

- #define [V2SUC\\_ALWAYS\\_OFF](#) 0x00U  
*Bits V2C is set to [V2C\\_OFF](#) (00) at power-up.*
- #define [V2SUC\\_ALWAYS\\_ON](#) 0x08U  
*Bits V2C is set to [V2C\\_ON\\_NSSRF](#) (11) at power-up.*

#### HVIO4 configuration control options (0x73:IO4SFC[2])

- #define [IO4SFC\\_STD\\_IO](#) 0x00U  
*Configure pin HVIO4 as a standard I/O pin.*
- #define [IO4SFC\\_LIMP\\_HOME](#) 0x04U  
*Enable HVIO4 limp home function.*

#### HVIO3 configuration control options (0x73:IO3SFC[1])

- #define [IO3SFC\\_STD\\_IO](#) 0x00U  
*Configure pin HVIO3 as a standard I/O pin.*
- #define [IO3SFC\\_LIMP\\_HOME](#) 0x02U  
*Enable HVIO3 limp home function.*



## HVIO2 configuration control options (0x73:IO2SFC[0])

- `#define IO2SFC_STD_IO 0x00U`  
*Configure pin HVIO2 as a standard I/O pin.*
- `#define IO2SFC_LIMP_HOME 0x01U`  
*Enable HVIO2 limp home function.*

## 16.124.2 Macro Definition Documentation

16.124.2.1 `#define IO2SFC_LIMP_HOME 0x01U`

Enable HVIO2 limp home function.

Definition at line 1309 of file `sbc_uja113x.h`.

16.124.2.2 `#define IO2SFC_STD_IO 0x00U`

Configure pin HVIO2 as a standard I/O pin.

Definition at line 1308 of file `sbc_uja113x.h`.

16.124.2.3 `#define IO3SFC_LIMP_HOME 0x02U`

Enable HVIO3 limp home function.

Definition at line 1302 of file `sbc_uja113x.h`.

16.124.2.4 `#define IO3SFC_STD_IO 0x00U`

Configure pin HVIO3 as a standard I/O pin.

Definition at line 1301 of file `sbc_uja113x.h`.

16.124.2.5 `#define IO4SFC_LIMP_HOME 0x04U`

Enable HVIO4 limp home function.

Definition at line 1295 of file `sbc_uja113x.h`.

16.124.2.6 `#define IO4SFC_STD_IO 0x00U`

Configure pin HVIO4 as a standard I/O pin.

Definition at line 1294 of file `sbc_uja113x.h`.

16.124.2.7 `#define RLC_PULSE_WIDTH_01_MS 0x30U`

$tw(rst) = 1 \text{ ms to } 1.5 \text{ ms}$

Definition at line 1280 of file `sbc_uja113x.h`.

16.124.2.8 `#define RLC_PULSE_WIDTH_03_05_MS 0x20U`

$tw(rst) = 3.6 \text{ ms to } 5 \text{ ms}$

Definition at line 1279 of file `sbc_uja113x.h`.

16.124.2.9 `#define RLC_PULSE_WIDTH_10_12_MS 0x10U`

$tw(rst) = 10 \text{ ms to } 12.5 \text{ ms}$

Definition at line 1278 of file `sbc_uja113x.h`.

16.124.2.10 `#define RLC_PULSE_WIDTH_20_25_MS 0x00U`

tw(rst) = 20 ms to 25 ms

Definition at line 1277 of file sbc\_uja113x.h.

16.124.2.11 `#define V2SUC_ALWAYS_OFF 0x00U`

Bits V2C is set to [V2C\\_OFF](#) (00) at power-up.

Definition at line 1287 of file sbc\_uja113x.h.

16.124.2.12 `#define V2SUC_ALWAYS_ON 0x08U`

Bits V2C is set to [V2C\\_ON\\_NSSRF](#) (11) at power-up.

Definition at line 1288 of file sbc\_uja113x.h.

## 16.125 Structural Core Self Test

Structural Core Self Test integration with S32 SDK

### General Information

- The SCST library provides tests to achieve the claimed diagnostic coverage (analytically estimated).
- The SCST library can be executed periodically at run time. This way, it contributes to a Single-Point Fault metric. The library preserves execution context of application and device configuration.
- The included tests cover most of the core instructions, as well as the tests targeting specific IP blocks of the core:
  - Core control logic (branch control, exception control);
  - Core data path including:
    - \* Register file and register multiplexing;
    - \* ALU, multiplier, divider, load/store, and other execution units;
    - \* SIMDSAT;
    - \* Instruction decoder, 16-Bit, 32-Bit;
- Interrupts can be enabled during execution of the most of the tests. SCST library provides its own interrupt vector table and wrappers for interrupt service routines, which in case of unexpected for the library interrupt, forwards it to the corresponding interrupt handler of the OS / user application. SCST library supports nested interrupts without any limitations.
- SCST library can be compiled and linked with other SCST libraries (e.g. SCST library for Cortex A5 core) within the same application for which a single .elf file is generated.

### Note

This is just a brief description of the Structural Core Self Test Library, for more information please check the full library documentation found in [<SDK\\_Location>/lib/<CPU\\_Family>/SCST/User\\_Documentation/<CoreType>>\\_<CPU\\_Name>\\_SCST\\_User\\_Manual.pdf](#)

The library is provided in binary format, compiled using GCC and for evaluation purposes only. Please consult license.txt file for more information found in [<SDK\\_Location>/lib/<CPU\\_Family>/SCST/license.txt](#)

The library was built with FPU(hard) enabled, so it must be enabled also in the application which will use it.

### How to use

To add SCST in your application you need to follow four steps:

- 1) Add sCST S32CT component into your project. The component will automatically add the required include paths and library files to the compilation.
- 2) Check that **\_m4\_scst** is added to Libraries under Linker build settings. If it is not present add a new entry with text **:lib\_m4\_scst.a**
- 3) Check that **"\${workspace\_loc:\${ProjName}}/SDK/lib/SCST/SCST/src/lib}"** is added to Library search paths under Linker build settings. If it is not present add a new entry with following text (include the quotes as well) **"\${workspace\_loc:\${ProjName}}/SDK/lib/SCST/SCST/src/lib"** or add the path to the folder where the library is located.
- 4) Add sCST code and data section to the linker file
  - Example(for GCC linker file):

```

.m4_scst :
{
    *(.m4_scst_test_code)
    *(.m4_scst_test_code_unprivileged)
    *(.m4_scst_test_code1_unprivileged)
    *(.m4_scst_test_shell_code)
    *(.m4_scst_rom_data)
    . = ALIGN(4);
    *(.m4_scst_exception_wrappers)
    *(.m4_scst_vector_table)
    . = ALIGN(4);
} > m_text

.m4_scst2 :
{
    __SCST_DATA_ROM = .;
    __scst_data_start__ = .;
    *(.m4_scst_test_shell_data)
    . = ALIGN(4);
    *(.m4_scst_ram_data)
    . = ALIGN(4);
    *(.m4_scst_ram_data_target0)
    . = ALIGN(4);
    *(.m4_scst_ram_data_target1)
    . = ALIGN(4);
    *(.m4_scst_ram_test_code)
    . = ALIGN(4);
    __scst_data_end__ = .;
} > m_data

```

- 5) Use the library API to execute the required tests.

You can use the sCST example as a practical implementation of the steps described above. SCST can not be used with RAM debug configuration due to RAM memory being too small.

## 16.126 Supply interrupt enable configuration options (0x1C)

### 16.126.1 Detailed Description

This section lists all the possible options of 0x1C register related to:

- SMPS status interrupt
- Battery monitor overvoltage interrupt
- Battery monitor undervoltage interrupt
- VEXT overvoltage interrupt
- VEXT undervoltage interrupt
- V1 undervoltage interrupt

See also

[UJA113X\\_SetSupplyInterrupts\(\)](#)

#### SMPS status interrupt configuration options (0x1C:SMPSSIE[5])

- #define [SMPSSIE\\_DISABLE](#) 0x00U  
*Disable SMPS status interrupt.*
- #define [SMPSSIE\\_ENABLE](#) 0x20U  
*Enable SMPS status interrupt.*

#### Battery monitor overvoltage interrupt configuration options (0x1C:BMOIE[4])

- #define [BMOIE\\_DISABLE](#) 0x00U  
*Disable battery monitor overvoltage interrupt.*
- #define [BMOIE\\_ENABLE](#) 0x10U  
*Enable battery monitor overvoltage interrupt.*

#### Battery monitor undervoltage interrupt configuration options (0x1C:BMUIE[3])

- #define [BMUIE\\_DISABLE](#) 0x00U  
*Disable battery monitor undervoltage interrupt.*
- #define [BMUIE\\_ENABLE](#) 0x08U  
*Enable battery monitor undervoltage interrupt.*

#### VEXT overvoltage interrupt configuration options (0x1C:VEXTOIE[2])

- #define [VEXTOIE\\_DISABLE](#) 0x00U  
*Disable VEXT overvoltage interrupt.*
- #define [VEXTOIE\\_ENABLE](#) 0x04U  
*Enable VEXT overvoltage interrupt.*

#### VEXT undervoltage interrupt configuration options (0x1C:VEXTUIE[1])

- #define [VEXTUIE\\_DISABLE](#) 0x00U  
*Disable VEXT undervoltage interrupt.*
- #define [VEXTUIE\\_ENABLE](#) 0x02U  
*Enable VEXT undervoltage interrupt.*

## V1UIE undervoltage interrupt configuration options (0x1C:V1UIE[0])

- `#define V1UIE_DISABLE 0x00U`  
*Disable V1 undervoltage interrupt.*
- `#define V1UIE_ENABLE 0x01U`  
*Enable V1 undervoltage interrupt.*

### 16.126.2 Macro Definition Documentation

#### 16.126.2.1 `#define BMOIE_DISABLE 0x00U`

Disable battery monitor overvoltage interrupt.

Definition at line 541 of file `sbc_uja113x.h`.

#### 16.126.2.2 `#define BMOIE_ENABLE 0x10U`

Enable battery monitor overvoltage interrupt.

Definition at line 542 of file `sbc_uja113x.h`.

#### 16.126.2.3 `#define BMUIE_DISABLE 0x00U`

Disable battery monitor undervoltage interrupt.

Definition at line 548 of file `sbc_uja113x.h`.

#### 16.126.2.4 `#define BMUIE_ENABLE 0x08U`

Enable battery monitor undervoltage interrupt.

Definition at line 549 of file `sbc_uja113x.h`.

#### 16.126.2.5 `#define SMPSSIE_DISABLE 0x00U`

Disable SMPS status interrupt.

Definition at line 534 of file `sbc_uja113x.h`.

#### 16.126.2.6 `#define SMPSSIE_ENABLE 0x20U`

Enable SMPS status interrupt.

Definition at line 535 of file `sbc_uja113x.h`.

#### 16.126.2.7 `#define V1UIE_DISABLE 0x00U`

Disable V1 undervoltage interrupt.

Definition at line 569 of file `sbc_uja113x.h`.

#### 16.126.2.8 `#define V1UIE_ENABLE 0x01U`

Enable V1 undervoltage interrupt.

Definition at line 570 of file `sbc_uja113x.h`.

#### 16.126.2.9 `#define VEXTOIE_DISABLE 0x00U`

Disable VEXT overvoltage interrupt.

Definition at line 555 of file `sbc_uja113x.h`.

16.126.2.10 `#define VEXTOIE_ENABLE 0x04U`

Enable VEXT overvoltage interrupt.

Definition at line 556 of file sbc\_uja113x.h.

16.126.2.11 `#define VEXTUIE_DISABLE 0x00U`

Disable VEXT undervoltage interrupt.

Definition at line 562 of file sbc\_uja113x.h.

16.126.2.12 `#define VEXTUIE_ENABLE 0x02U`

Enable VEXT undervoltage interrupt.

Definition at line 563 of file sbc\_uja113x.h.

## 16.127 System Basis Chip Driver (SBC) - UJA116xA Family

### 16.127.1 Detailed Description

System Basis Chip driver is a middleware driver for SBC settings and control.

#### Hardware background

The UJA116xA is a mini high-speed CAN System Basis Chip (SBC) containing an ISO 11898-2:201x compliant HS-CAN transceiver and an integrated 5 V or 3.3 V 250 mA scalable supply (V1) for a microcontroller and/or other loads. It also features a watchdog and a Serial Peripheral Interface (SPI). The UJA116xA can be operated in very low-current Standby and Sleep modes with bus and local wake-up capability. The UJA1169A comes in six variants. The UJA1169ATK, UJA1169ATK/F, UJA1169ATK/X and UJA1169ATK/X/F contain a 5 V regulator (V1). V1 is a 3.3 V regulator in the UJA1169ATK/3 and the UJA1169ATK/F/3. The UJA1169ATK, UJA1169ATK/F, UJA1169ATK/3 and UJA1169ATK/F/3 variants feature a second on-board 5 V regulator (V2) that supplies the internal CAN transceiver and can also be used to supply additional on-board hardware. The UJA1169ATK/X and UJA1169ATK/X/F are equipped with a 5 V supply (VEXT) for off-board components. VEXT is short-circuit proof to the battery, ground and negative voltages. The integrated CAN transceiver is supplied internally via V1, in parallel with the microcontroller. The UJA1168 comes in four variants. The UJA1168ATK, UJA1168ATK/FD, UJA1168ATK/VX and UJA1168ATK/VX/FD contain a 5 V regulator (V1). The UJA1168ATK and UJA1168ATK/FD versions contain a battery-related high-voltage output (INH) for controlling an external voltage regulator, while the UJA1168ATK/VX and UJA1168ATK/VX/FD are equipped with a 5 V sensor supply (VEXT). The UJA1169xx/F and UJA1168xx/FD variants support ISO 11898 compliant CAN partial networking with a selective wake-up function incorporating CAN FD-passive. CAN FD-passive is a feature that allows CAN FD bus traffic to be ignored in Sleep/Standby mode. CAN FD-passive partial networking is the perfect fit for networks that support both CAN FD and classic CAN communications. It allows normal CAN controllers that do not need to communicate CAN FD messages to remain in partial networking Sleep/Standby mode during CAN FD communication without generating bus errors. The UJA116xA implements the standard CAN physical layer as defined in the current ISO11898 standard (-2:2003, -5:2007, -6:2013). A dedicated LIMP output pin is provided to flag system failures on UJA1169 variants. A number of configuration settings are stored in non-volatile memory. This arrangement makes it possible to configure the power-on and limp-home behavior of the UJA116xA to meet the requirements of different applications.

#### How to use SBC driver in your application

In order to set up SBC device the user needs to configure `sbc_int_config_t` structure in which are included following structures: `sbc_regulator_ctr_t`, `sbc_wtdog_ctr_t`, `sbc_mode_mc_t`, `sbc_fail_safe_lhc_t`, `sbc_sys_evnt_t`, `sbc_lock_t`, `sbc_can_conf_t`, `sbc_wake_t`. These nested structures correspond to individual registers. The `sbc_int_config_t` structure is passed as a parameter to `Init` function to initialize SBC device. The rest of the functions are related to individual registers.

#### Initialization

The `SBC_InitDriver` function takes a parameter which is an instance of SPI used for communication with UJA116xA. The `SBC_InitDevice` function is responsible for setting up the UJA116xA, according to user configuration data which is passed as parameter. The `SBC_InitDevice` function configures all SBC registers except factories configuration set up in non volatile memory, (Start up control and SBC configuration register.) The `SBC_InitDevice` function transitions the SBC to standby mode, where the configuration is performed, and then to a mode selected in the main configuration structure. Before the transition to standby mode all event capture registers are cleared. In order to read the pre-reset or wake-up events, use `SBC_GetEventsStatus` between `SBC_InitDriver` and `SBC_InitDevice`.

#### Mode transition

`SBC_SetMode` performs software transition from one mode to another. The transition is achieved by writing to mode control register. The event capture registers are cleared before device is moved to standby and sleep mode.

#### Writing to registers

In order to write to registers, there are several methods dedicated to some specific registers. These methods (names starting with `SBC_Set`) take a value or a pointer to structure containing values to be written to particular



registers as a parameter. Besides these methods there is also a method `SBC_DataTransfer` which is common to reading and writing to all registers. It takes three parameters. The first one is an address of a register to be written. Addresses of registers are defined in `sbc_register_t` enum. The second argument is pointer to a value which should be sent to a register. The last argument is used for register reading only and its value is unused in this case. NULL pointer is used when parameter is unused.

#### Reading from registers

Content of a register is read by method `SBC_DataTransfer`, which provides both reading and writing to all registers. This method has three arguments. The first one is an address of a register to be read from, the third one is a pointer to a variable where the content of a register will be stored. Second argument is used for the register writing only and it should be NULL in this case. Addresses of registers are defined in enum `sbc_register_t`. Several methods to reading specific control and status register are available similarly to the register writing. Their names start with `SBC_Get`.

#### Reading status registers

Content of status register can be read by method `SBC_DataTransfer` or using appropriate function which starts with `SBC_Get` and finishes with `Status`. Event capture registers must be cleared using `SBC_CleanEvents` by setting to 1 appropriate status. For clear all events set all statuses to 1 or reading all event capture statuses using `SBC_GetEventsStatus` before.

There are several functions which read status and store it to structure. The Table 3 summarize which function reads appropriate status register.

Function name	Status register
<code>SBC_GetMainStatus</code>	Main status, Watchdog status
<code>SBC_GetSupplyStatus</code>	V2/VEXT status, V1 status
<code>SBC_GetCanStatus</code>	CAN transceiver status, CAN partial networking error, CAN partial networking status, CAN oscillator status, CAN-bus silence status, VCAN status, CAN failure status
<code>SBC_GetWakeStatus</code>	WAKE pin status
<code>SBC_GetEventsStatus</code>	Global event status, System event status, Supply event status, Transceiver event status, WAKE pin event status
<code>SBC_GetAllStatus</code>	Read all statuses from this table

#### Reading and writing non-volatile SBC configuration

The UJA116xA contains Multiple Time Programmable Non-Volatile (MTPNV) memory cells that allow some of the default device settings to be reconfigured. This non-volatile memory has limited write access. Programming of the NVM registers is performed in two steps. First, the required values are written. In the second step, reprogramming is confirmed by writing the correct CRC value to the MTPNV CRC control register. This memory is accessed by `SBC_GetFactoriesSettings` and `SBC_ChangeFactoriesSettings` methods. The only parameter is a pointer to `sbc_factories_conf_t` data structure, which should be written to NTPNVM or where should be stored data read out from the NTPNVM. If the device has been programmed previously, the factory presets may need to be restored before reprogramming can begin. When the factory presets have been restored successfully, a system reset is generated automatically and UJA116xA switches back to Forced Normal mode. If `SBC_ChangeFactoriesSettings` method returns an error "SBC\_UJA\_NVN\_ERROR" it means device was preconfigured from default settings and it is not possible to write to non-volatile memory. Restore factory preset values is needed. Factory preset values are restored if the following conditions apply continuously for at least `td(MTPNV)` during battery power-up: • pin RSTN is held LOW • CANH is pulled up to VBAT • CANL is pulled down to GND Now `SBC_ChangeFactoriesSettings` can be used for change factory preset values to custom configuration.

#### Error tracking

If an error during the R/W operations to UJA116xA registers occurs, the driver keeps track of it. If a method returns status different from `STATUS_SUCCESS` the status represents the type of error from `sbc_status_t` enum.

### Example code snippets (for FRDM PK144-Q100 freedom board).

#### Initialization example

This example source code snippet shows how to initialize SPI and the SBC. The SPI instance used in this example is LPSPICOM1. The following structures are generated by the configuration tool or need to be created by the user: lpspiCom1\_MasterConfig0, lpspiCom1State, sbc\_uja116x\_InitConfig0.

```
int main(void)
{
    ...

    sbc_status_t status = STATUS_SUCCESS;

    /* Initialization clocks. */
    ...

    /* Initialize pins. */
    ...

    /* Initialize LPSPI. */
    LPSPI_DRV_MasterInit(LPSPICOM1, &lpspiCom1State, &lpspiCom1_MasterConfig0);

    /* Initialize SBC. */
    status = SBC_InitDriver(LPSPICOM1);
    status |= SBC_InitDevice(&sbc_uja116x_InitConfig0);

    if(status != STATUS_SUCCESS)
    {
        /* Do something here. */
    }

    ...
}
```

#### ### Write to Regulator control registers example ###

This example source code snippet shows how to configure Regulator control register. Power distribution control (PDC), V2/VEXT configuration (V2C/ VETXT), V1 reset threshold can be configured by writing to Regulator Control register. Note: PDC can be set for UJA1169 variants (not UJA1168), V2 can be set for models UJA1169ATK, UJA1169ATK/3, UJA1169ATK/F and UJA1169ATK/F/3, VEXT can be set for models UJA1169ATK/X and UJA1169ATK/X/F. For more info read function description.

```
int main(void)
{
    ...

    sbc_status_t status = STATUS_SUCCESS;
    sbc_regulator_ctr_t regulator;
    regulator.regulator.pdc = SBC_UJA_REGULATOR_PDC_HV;
    regulator.regulator.v2c = SBC_UJA_REGULATOR_V2C_OFF;
    regulator.regulator.v1rtc = SBC_UJA_REGULATOR_V1RTC_80;

    regulator.supplyEvt.v2oe = SBC_UJA_SUPPLY_EVT_V2OE_EN;
    regulator.supplyEvt.v2ue = SBC_UJA_SUPPLY_EVT_V2UE_EN;
    regulator.supplyEvt.v1ue = SBC_UJA_SUPPLY_EVT_V1UE_DIS;

    status = SBC_SetVreg(&regulator);

    if(status != STATUS_SUCCESS)
    {
        /* Do something here. */
    }

    ...
}
```

#### Read from Regulator control registers example

This example source code snippet shows how to read from Regulator control registers. Reading Regulator control register gives information about Power distribution control (PDC), V2/VEXT configuration (V2C/ VETXT), V1 reset threshold current configuration. Using this method can be useful for check if the Regulator control register is configured correctly. For more info read function description.

```
int main(void)
{
```

```

...

sbc_status_t status = STATUS_SUCCESS;
sbc_regulator_ctr_t regulator;

status = SBC_GetVreg(&regulator);

if(status == STATUS_SUCCESS)
{
    if(regulator.supplyEvt.v2oe ==
        SBC_UJA_SUPPLY_EVT_V2OE_EN)
    {
        /* Do something here. */
    }
}

...
}

```

### Reading all device status example

This example source code snippet shows how to read all SBC device statuses in one function. Variable allStatuses contains these registers: Main status register, Watchdog status register, Supply voltage status register, Transceiver status register, WAKE pin status register, Event capture registers. For more info read function description.

```

int main(void)
{
    ...

    sbc_status_t status = STATUS_SUCCESS;
    sbc_status_group_t allStatuses;

    while(1) {

        status = SBC_GetAllStatus(&allStatuses);

        if(status == STATUS_SUCCESS)
        {

            if(allStatuses.trans.cbss == SBC_UJA_TRANS_STAT_CBSS_ACT)
            {
                /* Do something here. */
            }

            if(allStatuses.supply.vls == SBC_UJA_SUPPLY_STAT_VLS_VAB)
            {
                /* Do something here. */
            }

            ...

            /* Periodically feed watchdog (anytime in watchdog period in case of timeout watchdog mode). */
            SBC_FeedWatchdog();
        }
    }
}

```

### Reading Transceiver device status example

This example source code snippet shows how to read Transceiver device status from SBC. It contains CAN transceiver status, CAN partial networking error, CAN partial networking status, CAN oscillator status, CAN-bus silence status, VCAN status, CAN failure status. For more info read function description. Note similar approach can be used for reading other status using different SBC\_Get\*Status.

```

int main(void)
{
    ...

    sbc_status_t status = STATUS_SUCCESS;
    sbc_trans_stat_t transStatus;

    while(1) {

        status = SBC_GetCanStatus(&transStatus);

        if(status == STATUS_SUCCESS)
        {

            if(transStatus.cbss == SBC_UJA_TRANS_STAT_CBSS_ACT)

```

```

    {
        /* Do something here. */
    }

    ...

    /* Periodically feed watchdog (anytime in watchdog period in case of timeout watchdog mode). */
    SBC_FeedWatchdog();
}
}
}

```

### Change factories settings

This example source code snippet shows how to change factory preset value of non-volatile memory. Device must be set to factory preset. For more info read function description.

```

int main(void)
{
    ...

    sbc_status_t status = STATUS_SUCCESS;
    sbc_factories_conf_t factories;

    status = SBC_GetFactoriesSettings(&factories);

    factories.control.fnmc = SBC_UJA_SBC_SDMC_EN;
    factories.control.sdmc = SBC_UJA_SBC_SDMC_DIS;
    factories.startUp.rlc = SBC_UJA_START_UP_RLC_20_25p0;

    if(status == STATUS_SUCCESS)
    {
        status = SBC_ChangeFactoriesSettings(&factories);
    }

    if(status != STATUS_SUCCESS)
    {
        /* Do something here. */
    }
}

```

### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```

${S32SDK_PATH}\middleware\sbc\sbc_uja116x\source\sbc_uja116x_driver.c

```

#### Include path

The following paths need to be added to the include path of the toolchain:

```

${S32SDK_PATH}\middleware\sbc\sbc_uja116x\include

```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

[Low Power Serial Peripheral Interface \(LPSPI\) OS Interface \(OSIF\)](#)

#### Modules

- [UJA116xA SBC Driver](#)

## 16.128 System Basis Chip library (SBC) - UJA113x Family

### 16.128.1 Detailed Description

System Basis Chip library for configuration and control.

### 16.128.2 Introduction

The SBC is quite complex device with many features (CAN/CAN FD & LIN transceivers, Buck/boost driver, watch-dog etc.), several modes (Normal, Standby, Sleep, Forced normal mode, Software Development mode etc.) and provides interrupt/polling based error reporting mechanism to the Host MCU. Different applications require different functionalities ranging from a single feature usage to a complete SBC functionality. From this perspective it's not possible to provide a driver, which would satisfy the needs of all the applications. The driver would become too complicated with a large code footprint and the user will use it as a guide how to create his own application at the end instead.

From the abovementioned reasons the library is not a driver and should not be understood that way. It's a library which in conjunction with the S32 SDK provides a low level API allowing the user fast product evaluation in the first step. After becoming familiar with the device, the API can be used as a basis for a more complex application, where the user determines, whether interrupt based error handling or polling is sufficient, which errors to handle, how and when to switch between modes etc.

From programming perspective the chip is divided into following register groups:

- [01. Primary control](#) registers
- [02. Supply control](#) registers
- [03. Transceiver control](#) registers
- [04. HVIO bank 0 & 1 control](#) registers
- [05. Timer control](#) registers
- [06. Interrupts](#)
- [07. MTPNV and ID](#) registers

Each group represents a set of registers for configuring different part/functionality of the SBC. For instance the Primary control registers can be used to configure the Watchdog, SBC mode, Fail-safe behavior, General-purpose user memory etc. The API is designed that way to facilitate the users chip configuration by allowing the user to configure the chip per functionality.

For library internals description and how to start using this library, proceed to [00. Library internals](#) section.

#### Modules

- [00. Library internals](#)
- [01. Primary control](#)
- [02. Supply control](#)
- [03. Transceiver control](#)
- [04. HVIO bank 0 & 1 control](#)
- [05. Timer control](#)
- [06. Interrupts](#)
- [07. MTPNV and ID](#)

## 16.129 System interrupt enable configuration options (0x04)

### 16.129.1 Detailed Description

This section lists all the possible options of 0x04 register related to:

- Overtemperature interrupt
- SPI failure interrupt

See also

[UJA113X\\_SetSystemInterrupts\(\)](#)

### Overtemperature warning interrupt configuration options (0x04:OTWIE[2])

- #define [OTWIE\\_DISABLE](#) 0x00U  
*Disable overtemperature warning interrupt.*
- #define [OTWIE\\_ENABLE](#) 0x04U  
*Enable overtemperature warning interrupt.*

### SPI failure interrupt configuration options (0x04:SPIFIE[1])

- #define [SPIFIE\\_DISABLE](#) 0x00U  
*Disable SPI failure interrupt.*
- #define [SPIFIE\\_ENABLE](#) 0x02U  
*Enable SPI failure interrupt.*

### 16.129.2 Macro Definition Documentation

#### 16.129.2.1 #define OTWIE\_DISABLE 0x00U

Disable overtemperature warning interrupt.

Definition at line 343 of file sbc\_uja113x.h.

#### 16.129.2.2 #define OTWIE\_ENABLE 0x04U

Enable overtemperature warning interrupt.

Definition at line 344 of file sbc\_uja113x.h.

#### 16.129.2.3 #define SPIFIE\_DISABLE 0x00U

Disable SPI failure interrupt.

Definition at line 350 of file sbc\_uja113x.h.

#### 16.129.2.4 #define SPIFIE\_ENABLE 0x02U

Enable SPI failure interrupt.

Definition at line 351 of file sbc\_uja113x.h.

## 16.130 TRGMUX Driver

### 16.130.1 Detailed Description

Trigger MUX Control Peripheral Driver. The TRGMUX introduces an extremely flexible methodology for connecting various trigger sources to multiple pins/peripherals.

The S32 SDK provides Peripheral Drivers for the Trigger MUX Control (TRGMUX) module of S32 SDK devices.

#### Overview

This section describes the programming interface of the TRGMUX driver. The TRGMUX driver configures the TRGMUX (Trigger Mux Control). The Trigger MUX module allows software to configure the trigger inputs for various peripherals.

#### TRGMUX Driver model building

TRGMUX can be seen as a collection of muxes, each mux allowing to select one output from a list of input signals that are common to all muxes. The TRGMUX registers are identical as structure and all bitfields can be read/written using the TRGMUX driver API.

#### TRGMUX Initialization

The [TRGMUX\\_DRV\\_Init\(\)](#) function is used to initialize the TRGMUX IP. The function receives as parameter a pointer to the [trgmux\\_user\\_config\\_t](#) structure. This structure contains a variable number of mappings between a trgmux trigger source and a trgmux target modules.

#### Integration guideline

##### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\drivers\src\trgmux\trgmux_driver.c
${S32SDK_PATH}\platform\drivers\src\trgmux\trgmux_hw_access.c
```

##### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc\
```

##### Compile symbols

No special symbols are required for this component

##### Dependencies

There are no dependencies with other components

#### TRGMUX API

After initialization, the driver allows the reconfiguration of the source trigger for a given target module using [TRGMUX\\_DRV\\_SetTrigSourceForTargetModule\(\)](#). Also, by using [TRGMUX\\_DRV\\_SetLockForTargetModule\(\)](#), a given target module can be locked, such that it cannot be updated until a reset.

#### Data Structures

- struct [trgmux\\_inout\\_mapping\\_config\\_t](#)  
Configuration structure for pairing source triggers with target modules. [More...](#)
- struct [trgmux\\_user\\_config\\_t](#)

User configuration structure for the TRGMUX driver. [More...](#)

## Typedefs

- typedef enum trgmux\_trigger\_source\_e [trgmux\\_trigger\\_source\\_t](#)  
*Enumeration for trigger source module of TRGMUX.*
- typedef enum trgmux\_target\_module\_e [trgmux\\_target\\_module\\_t](#)  
*Enumeration for target module of TRGMUX.*

## Functions

- status\_t [TRGMUX\\_DRV\\_Init](#) (const uint32\_t instance, const [trgmux\\_user\\_config\\_t](#) \*const trgmuxUserConfig)  
*Initialize a TRGMUX instance for operation.*
- status\_t [TRGMUX\\_DRV\\_Deinit](#) (const uint32\_t instance)  
*Reset to default values the source triggers corresponding to all target modules, if none of the target modules is locked.*
- status\_t [TRGMUX\\_DRV\\_SetTrigSourceForTargetModule](#) (const uint32\_t instance, const [trgmux\\_trigger\\_source\\_t](#) triggerSource, const [trgmux\\_target\\_module\\_t](#) targetModule)  
*Configure a source trigger for a selected target module.*
- [trgmux\\_trigger\\_source\\_t](#) [TRGMUX\\_DRV\\_GetTrigSourceForTargetModule](#) (const uint32\_t instance, const [trgmux\\_target\\_module\\_t](#) targetModule)  
*Get the source trigger configured for a target module.*
- void [TRGMUX\\_DRV\\_SetLockForTargetModule](#) (const uint32\_t instance, const [trgmux\\_target\\_module\\_t](#) targetModule)  
*Locks the TRGMUX register of a target module.*
- bool [TRGMUX\\_DRV\\_GetLockForTargetModule](#) (const uint32\_t instance, const [trgmux\\_target\\_module\\_t](#) targetModule)  
*Get the Lock bit status of the TRGMUX register of a target module.*
- void [TRGMUX\\_DRV\\_GenSWTrigger](#) (const uint32\_t instance)  
*Generate software triggers.*

## 16.130.2 Data Structure Documentation

### 16.130.2.1 struct trgmux\_inout\_mapping\_config\_t

Configuration structure for pairing source triggers with target modules.

Use an instance of this structure to define a TRGMUX link between a trigger source and a target module. This structure is used by the user configuration structure.

Implements : [trgmux\\_inout\\_mapping\\_config\\_t\\_Class](#)

Definition at line 88 of file [trgmux\\_driver.h](#).

## Data Fields

- [trgmux\\_trigger\\_source\\_t](#) triggerSource
- [trgmux\\_target\\_module\\_t](#) targetModule
- bool lockTargetModuleReg

## Field Documentation

### 16.130.2.1.1 bool lockTargetModuleReg

if true, the LOCK bit of the target module register will be set by [TRGMUX\\_DRV\\_INIT\(\)](#), after the current mapping is configured

Definition at line 92 of file [trgmux\\_driver.h](#).



**16.130.2.1.2 trgmux\_target\_module\_t targetModule**

selects one of the TRGMUX target modules

Definition at line 91 of file trgmux\_driver.h.

**16.130.2.1.3 trgmux\_trigger\_source\_t triggerSource**

selects one of the TRGMUX trigger sources

Definition at line 90 of file trgmux\_driver.h.

**16.130.2.2 struct trgmux\_user\_config\_t**

User configuration structure for the TRGMUX driver.

Use an instance of this structure with the [TRGMUX\\_DRV\\_Init\(\)](#) function. This enables configuration of TRGMUX with the user defined mappings between inputs (source triggers) and outputs (target modules), via a single function call.

Implements : trgmux\_user\_config\_t\_Class

Definition at line 104 of file trgmux\_driver.h.

**Data Fields**

- [uint8\\_t numInOutMappingConfigs](#)
- [const trgmux\\_inout\\_mapping\\_config\\_t \\* inOutMappingConfig](#)

**Field Documentation****16.130.2.2.1 const trgmux\_inout\_mapping\_config\_t\* inOutMappingConfig**

pointer to array of in-out mapping structures

Definition at line 107 of file trgmux\_driver.h.

**16.130.2.2.2 uint8\_t numInOutMappingConfigs**

number of in-out mappings defined in TRGMUX configuration

Definition at line 106 of file trgmux\_driver.h.

**16.130.3 Typedef Documentation****16.130.3.1 typedef enum trgmux\_target\_module\_e trgmux\_target\_module\_t**

Enumeration for target module of TRGMUX.

Describes all possible outputs (target modules) of the TRGMUX IP This enumeration depends on the supported instances in device

Implements : trgmux\_target\_module\_t\_Class

Definition at line 78 of file trgmux\_driver.h.

**16.130.3.2 typedef enum trgmux\_trigger\_source\_e trgmux\_trigger\_source\_t**

Enumeration for trigger source module of TRGMUX.

Describes all possible inputs (trigger sources) of the TRGMUX IP This enumeration depends on the supported instances in device

Implements : trgmux\_trigger\_source\_t\_Class

Definition at line 68 of file trgmux\_driver.h.

#### 16.130.4 Function Documentation

##### 16.130.4.1 `status_t TRGMUX_DRV_Deinit ( const uint32_t instance )`

Reset to default values the source triggers corresponding to all target modules, if none of the target modules is locked.

###### Parameters

<code>in</code>	<code>instance</code>	The TRGMUX instance number.
-----------------	-----------------------	-----------------------------

###### Returns

Execution status:

STATUS\_SUCCESS

STATUS\_ERROR - if at least one of the target module register is locked.

Definition at line 114 of file `trgmux_driver.c`.

##### 16.130.4.2 `void TRGMUX_DRV_GenSWTrigger ( const uint32_t instance )`

Generate software triggers.

This function uses a SIM register in order to generate a software triggers to the target peripherals selected in TRGMUX

###### Parameters

<code>param[in]</code>	<code>instance</code>	The TRGMUX instance number.
------------------------	-----------------------	-----------------------------

Definition at line 221 of file `trgmux_driver.c`.

##### 16.130.4.3 `bool TRGMUX_DRV_GetLockForTargetModule ( const uint32_t instance, const trgmux_target_module_t targetModule )`

Get the Lock bit status of the TRGMUX register of a target module.

This function gets the value of the LK bit from the TRGMUX register corresponding to the selected target module.

###### Parameters

<code>in</code>	<code>instance</code>	The TRGMUX instance number.
<code>in</code>	<code>targetModule</code>	One of the values in the <code>trgmux_target_module_t</code> enumeration

###### Returns

true - if the selected targetModule register is locked

false - if the selected targetModule register is not locked

Definition at line 203 of file `trgmux_driver.c`.

##### 16.130.4.4 `trgmux_trigger_source_t TRGMUX_DRV_GetTrigSourceForTargetModule ( const uint32_t instance, const trgmux_target_module_t targetModule )`

Get the source trigger configured for a target module.

This function returns the TRGMUX source trigger linked to a selected target module.

###### Parameters

in	<i>instance</i>	The TRGMUX instance number.
in	<i>targetModule</i>	One of the values in the <code>trgmux_target_module_t</code> enumeration.

**Returns**

Enum value corresponding to the trigger source configured for the selected target module.

Definition at line 168 of file `trgmux_driver.c`.

#### 16.130.4.5 `status_t TRGMUX_DRV_Init ( const uint32_t instance, const trgmux_user_config_t *const trgmuxUserConfig )`

Initialize a TRGMUX instance for operation.

This function first resets the source triggers of all TRGMUX target modules to their default values, then configures the TRGMUX with all the user defined in-out mappings. If at least one of the target modules is locked, the function will not change any of the TRGMUX target modules and return error code. This example shows how to set up the `trgmux_user_config_t` parameters and how to call the `TRGMUX_DRV_Init()` function with the required parameters:

```

1 trgmux_user_config_t          trgmuxConfig;
2 trgmux_inout_mapping_config_t trgmuxInOutMappingConfig[] =
3 {
4     {TRGMUX_TRIG_SOURCE_TRGMUX_IN9,    TRGMUX_TARGET_MODULE_DMA_CH0,    false},
5     {TRGMUX_TRIG_SOURCE_FTM1_EXT_TRIG, TRGMUX_TARGET_MODULE_TRGMUX_OUT4, true}
6 };
7
8 trgmuxConfig.numInOutMappingConfigs = 2;
9 trgmuxConfig.inOutMappingConfig    = trgmuxInOutMappingConfig;
10
11 TRGMUX_DRV_Init(instance, &trgmuxConfig);

```

**Parameters**

in	<i>instance</i>	The TRGMUX instance number.
in	<i>trgmuxUserConfig</i>	Pointer to the user configuration structure.

**Returns**

Execution status:

`STATUS_SUCCESS`

`STATUS_ERROR` - if at least one of the target module register is locked.

Definition at line 71 of file `trgmux_driver.c`.

#### 16.130.4.6 `void TRGMUX_DRV_SetLockForTargetModule ( const uint32_t instance, const trgmux_target_module_t targetModule )`

Locks the TRGMUX register of a target module.

This function sets the LK bit of the TRGMUX register corresponding to the selected target module. Please note that some TRGMUX registers can contain up to 4 SEL bitfields, meaning that these registers can be used to configure up to 4 target modules independently. Because the LK bit is only one per register, the configuration of all target modules referred from that register will be locked.

**Parameters**

in	<i>instance</i>	The TRGMUX instance number.
in	<i>targetModule</i>	One of the values in the <code>trgmux_target_module_t</code> enumeration

Definition at line 185 of file `trgmux_driver.c`.

#### 16.130.4.7 `status_t TRGMUX_DRV_SetTrigSourceForTargetModule ( const uint32_t instance, const trgmux_trigger_source_t triggerSource, const trgmux_target_module_t targetModule )`

Configure a source trigger for a selected target module.

This function configures a TRGMUX link between a source trigger and a target module, if the requested target module is not locked.

**Parameters**

in	<i>instance</i>	The TRGMUX instance number.
in	<i>triggerSource</i>	One of the values in the <code>trgmux_trigger_source_t</code> enumeration
in	<i>targetModule</i>	One of the values in the <code>trgmux_target_module_t</code> enumeration

**Returns**

Execution status:

STATUS\_SUCCESS

STATUS\_ERROR - if requested target module is locked

Definition at line 135 of file `trgmux_driver.c`.

## 16.131 Timing - Peripheral Abstraction Layer (TIMING PAL)

### 16.131.1 Detailed Description

The S32 SDK provides a Peripheral Abstraction Layer for timer modules of S32 SDK devices.

The TIMING PAL driver allows to generate period event. It was designed to be portable across all platforms and IPs which support LPIT, PIT, LPTMR, FTM, STM.

#### How to integrate TIMING PAL in your application

Unlike the other drivers, TIMING PAL modules need to include a configuration file named `timing_pal_cfg.h`, which allows the user to specify which IPs are used. The following code example shows how to configure one instance for each available TIMING IPs.

```
#ifndef TIMING_PAL_CFG_H
#define TIMING_PAL_CFG_H

/* Define which IP instance will be used in current project */
#define TIMING_OVER_LPIT
#define TIMING_OVER_FTM
#define TIMING_OVER_LPTMR

#endif /* TIMING_PAL_CFG_H */
```

The following table contains the matching between platforms and available IPs

LPIT PIT M CU	S32K116	S32K118	S32K142	S32K144	S32K146	S32K148	S32K142W	S32K144W	S32V234	S32R274	S32R372	MPC5748G	MPC5746C	MPC5744P
LPIT PIT M CU	YES	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	NO	NO	NO
LPIT PIT M R CU	YES	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	NO	NO	NO

F↔ T↔ M↔ _↔ TI↔ M↔ I↔ NG	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	NO	NO	NO	NO	NO
PI↔ T↔ _↔ TI↔ M↔ I↔ NG	NO	NO	NO	NO	NO	NO	NO	NO	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES
S↔ T↔ M↔ _↔ TI↔ M↔ I↔ NG	NO	NO	NO	NO	NO	NO	NO	NO	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES	Y↔ ES

#### Features

- Start timer channel counting with period in ticks function
- Generate one-shot or continuous notification(Event)
- Get elapsed time and remaining time functions
- Get tick resolution in engineering units (nanosecond, microsecond or millisecond)

#### Functionality

##### Initialization

In order to use the TIMING PAL driver it must be first initialized, using [TIMING\\_Init\(\)](#) function. Once initialized, it should be de-initialized before initialized again for the same TIMING module instance, using [TIMING\\_Deinit\(\)](#). The initialization function does the following operations:

- sets the clock source, clock prescaler (except LPIT, PIT\_TIMING)
- sets notification type and callback function of timer channel Different TIMING modules instances can function independently of each other.

##### Start/Stop a timer channel counting with new period

After initialization, a timer channel can be started by calling [TIMING\\_StartChannel](#) function. The input period unit is ticks, the max value of period depends on which timer is used for timing. The [TIMING\\_StartChannel](#) function can be called consecutively, it starts new period immediately but in case LPIT, PIT\_TIMING when timer channel is running, to abort the current timer channel period and start a timer channel period with a new value, the timer channel must be stopped and started again. A timer channel can be stop by calling [TIMING\\_StopChannel](#) function.

##### Get elapsed and remaining time

When a timer channel is running, the elapsed and remaining timer can be got by calling [TIMING\\_GetElapsed](#) and [TIMING\\_GetRemaining](#) function. The elapsed and remaining time in nanosecond, microsecond or millisecond is the result of this function multiplies by the result of the [TIMING\\_GetResolution](#).

## Important Notes

- Before using the TIMING PAL driver the module clock must be configured. Refer to Clock Manager for clock configuration.
- The driver enables the interrupts for the corresponding TIMING module, but any interrupt priority must be done by the application
- When the vector table is not in ram (**flash\_vector\_table** = 1):
  - INT\_SYS\_InstallHandler shall check if the function pointer provided as parameter for the new handler is already present in the vector table for the given IRQ number.
  - The user will be required to manually add the correct handlers in the startup files
- The board specific configurations must be done prior to driver calls
- Some features are not available for all TIMING IPs and incorrect parameters will be handled by DEV\_ASSERT
- Because of the driver code limit, when use FTM\_TIMING or STM\_TIMING the executing time of interrupt handler is about 4 microseconds, so the erroneous period is about 4 microsecond, should configure period enough to skip this erroneous period.
- In MPC574xG and MPC574xC devices, STM module has Errata e10200. If STM clock source is configured to use the FXOSC clock for the case application software reads the STM Count register(STM\_CNT) while the counter is running, the value returned may be incorrect. Note the default clock source for the STM is the FS80 (divided system clock) and this configuration is working properly. Consequently the value returned by the function STM\_DRV\_GetCounterValue() may be incorrect when clock source is set to FXOSC option. Thus, this issue impacts to the functions that call STM\_DRV\_GetCounterValue() function:
  - TIMING\_GetElapsed
  - TIMING\_GetRemaining
  - TIMING\_StartChannel only in case the timer is already started This issue should be taken care when using TIMING over STM.

## ## Example code ##

```

/* The timer channel number */
#define TIMER_CHANNEL 0U

/* The timer channel period by nanosecond */
#define TIMER_PERIOD_NANO 1000000000U

/* Counting variable */
uint32_t count = 0;

/* Callback function */
void My_Callback(void * data)
{
    (void) data;
    count = count + 1;
}

/* Configure timer channel */
timer_chan_config_t timing_pall_channelArray[] =
{
    {
        TIMER_CHANNEL,
        TIMER_CHAN_TYPE_CONTINUOUS,
        My_Callback,
        NULL
    }
};

/* Configure FTM clock source */
extension_ftm_for_timer_t timing_pall_ftmExtention =
{
    FTM_CLOCK_SOURCE_FIXEDCLK,
    FTM_CLOCK_DIVID_BY_1,
    0xFFFF
};

/* Configure TIMING instance */

```



```

timer_config_t timing_pall_InitConfig =
{
    timing_pall_channelArray,
    1,
    &timing_pall_ftmExtention
};

/* TIMING instance number structure */
timing_instance_t instance =
{
    TIMING_INST_TYPE_LPIT,
    0U
};

int main(void)
{
    uint64_t resolution;
    uint32_t elapsedTime;

    /* Initialize TIMING */
    TIMING_Init(&instance, &timing_pall_InitConfig);

    /* Get tick resolution in nanosecond */
    TIMING_GetResolution(&instance,
        TIMER_RESOLUTION_TYPE_NANOSECOND, &resolution);

    /* Start channel counting with period is 1 second */
    TIMING_StartChannel(&instance, TIMER_CHANNEL, (TIMER_PERIOD_NANO / resolution));
    ....
    /* Get elapsed time in ticks */
    elapsedTime = TIMING_GetElapsed(&instance, TIMER_CHANNEL);

    /* Get elapsed time in nanosecond */
    elapsedTime = elapsedTime * resolution;

    /* De-initialize TIMING */
    TIMING_Deinit(&instance);
}

```

### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```

${S32SDK_PATH}\platform\pal\src\timing\timing_pal.c
${S32SDK_PATH}\platform\pal\src\timing\timing_irq.c

```

#### Include path

The following paths need to be added to the include path of the toolchain:

```

${S32SDK_PATH}\platform\pal\inc\

```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

**Common:** [Clock Manager](#) [Interrupt Manager](#) ([Interrupt](#))

**S32K1xx:** [Low Power Interrupt Timer \(LPIT\)](#) [Low Power Timer \(LPTMR\)](#) [FlexTimer \(FTM\)](#)

**MPC574x** and **S32Rx7x:** [pit stm](#)

#### Data Structures

- struct [timer\\_chan\\_config\\_t](#)  
Structure to configure the channel timer notification. [More...](#)
- struct [timer\\_config\\_t](#)  
Timer configuration structure. [More...](#)

- struct [extension\\_lptmr\\_for\\_timer\\_t](#)  
*Defines the extension structure for the timer over LPTMR. [More...](#)*
- struct [extension\\_ftm\\_for\\_timer\\_t](#)  
*Defines the extension structure for the timer over FTM. [More...](#)*

## Enumerations

- enum [timer\\_resolution\\_type\\_t](#) { [TIMER\\_RESOLUTION\\_TYPE\\_NANOSECOND](#), [TIMER\\_RESOLUTION\\_TYPE\\_MICROSECOND](#), [TIMER\\_RESOLUTION\\_TYPE\\_MILLISECOND](#) }  
*Type options available for tick resolution.*
- enum [timer\\_chan\\_type\\_t](#) { [TIMER\\_CHAN\\_TYPE\\_CONTINUOUS](#), [TIMER\\_CHAN\\_TYPE\\_ONESHOT](#) }  
*Type options available for timer channel notification.*

## Functions

- status\_t [TIMING\\_Init](#) (const [timing\\_instance\\_t](#) \*const instance, const [timer\\_config\\_t](#) \*const config)  
*Initialize the timer instance and timer channels with value from input configuration structure.*
- void [TIMING\\_Deinit](#) (const [timing\\_instance\\_t](#) \*const instance)  
*De-initialize a timer instance.*
- void [TIMING\\_StartChannel](#) (const [timing\\_instance\\_t](#) \*const instance, const uint8\_t channel, const uint32\_t periodTicks)  
*Starts the timer channel counting.*
- void [TIMING\\_StopChannel](#) (const [timing\\_instance\\_t](#) \*const instance, const uint8\_t channel)  
*Stop the timer channel counting.*
- uint32\_t [TIMING\\_GetElapsed](#) (const [timing\\_instance\\_t](#) \*const instance, const uint8\_t channel)  
*Get elapsed ticks.*
- uint32\_t [TIMING\\_GetRemaining](#) (const [timing\\_instance\\_t](#) \*const instance, const uint8\_t channel)  
*Get remaining ticks.*
- void [TIMING\\_EnableNotification](#) (const [timing\\_instance\\_t](#) \*const instance, const uint8\_t channel)  
*Enable channel notifications.*
- void [TIMING\\_DisableNotification](#) (const [timing\\_instance\\_t](#) \*const instance, const uint8\_t channel)  
*Disable channel notifications.*
- status\_t [TIMING\\_GetResolution](#) (const [timing\\_instance\\_t](#) \*const instance, const [timer\\_resolution\\_type\\_t](#) type, uint64\_t \*const resolution)  
*Get tick resolution.*
- status\_t [TIMING\\_GetMaxPeriod](#) (const [timing\\_instance\\_t](#) \*const instance, const [timer\\_resolution\\_type\\_t](#) type, uint64\_t \*const maxPeriod)  
*Get max period in engineering units.*
- void [TIMING\\_InstallCallback](#) (const [timing\\_instance\\_t](#) \*const instance, const uint8\_t channel, const [timer\\_callback\\_t](#) callback, void \*const callbackParam)  
*Installs callback function for the timer channel.*

## 16.131.2 Data Structure Documentation

### 16.131.2.1 struct [timer\\_chan\\_config\\_t](#)

Structure to configure the channel timer notification.

This structure holds the configuration settings for the timer channel notification Implements : [timer\\_chan\\_config\\_t\\_Class](#)

Definition at line 77 of file [timing\\_pal.h](#).

**Data Fields**

- [uint8\\_t channel](#)
- [timer\\_chan\\_type\\_t chanType](#)
- [timer\\_callback\\_t callback](#)
- [void \\* callbackParam](#)

**Field Documentation****16.131.2.1.1 timer\_callback\_t callback**

Callback function called on notification

Definition at line 81 of file timing\_pal.h.

**16.131.2.1.2 void\* callbackParam**

Callback parameter pointer

Definition at line 82 of file timing\_pal.h.

**16.131.2.1.3 uint8\_t channel**

Channel number

Definition at line 79 of file timing\_pal.h.

**16.131.2.1.4 timer\_chan\_type\_t chanType**

Continuous or One-shot

Definition at line 80 of file timing\_pal.h.

**16.131.2.2 struct timer\_config\_t**

Timer configuration structure.

This structure holds the configuration settings for the timer Implements : timer\_config\_t\_Class

Definition at line 91 of file timing\_pal.h.

**Data Fields**

- [const timer\\_chan\\_config\\_t \\* chanConfigArray](#)
- [uint8\\_t numChan](#)
- [void \\* extension](#)

**Field Documentation****16.131.2.2.1 const timer\_chan\_config\_t\* chanConfigArray**

Channel configuration array

Definition at line 93 of file timing\_pal.h.

**16.131.2.2.2 void\* extension**

IP specific configuration structure

Definition at line 95 of file timing\_pal.h.

**16.131.2.2.3 uint8\_t numChan**

Number of elements in chanConfigArray

Definition at line 94 of file timing\_pal.h.

### 16.131.2.3 struct extension\_lptmr\_for\_timer\_t

Defines the extension structure for the timer over LPTMR.

Part of LPTMR configuration structure Implements : extension\_lptmr\_for\_timer\_t\_Class

Definition at line 114 of file timing\_pal.h.

#### Data Fields

- [lptmr\\_clocksource\\_t clockSelect](#)
- [lptmr\\_prescaler\\_t prescaler](#)
- bool [bypassPrescaler](#)

#### Field Documentation

##### 16.131.2.3.1 bool bypassPrescaler

Enable/Disable prescaler bypass

Definition at line 118 of file timing\_pal.h.

##### 16.131.2.3.2 lptmr\_clocksource\_t clockSelect

LPTMR clock source selection

Definition at line 116 of file timing\_pal.h.

##### 16.131.2.3.3 lptmr\_prescaler\_t prescaler

Prescaler Selection

Definition at line 117 of file timing\_pal.h.

### 16.131.2.4 struct extension\_ftm\_for\_timer\_t

Defines the extension structure for the timer over FTM.

Part of FTM configuration structure Implements : extension\_ftm\_for\_timer\_t\_Class

Definition at line 129 of file timing\_pal.h.

#### Data Fields

- [ftm\\_clock\\_source\\_t clockSelect](#)
- [ftm\\_clock\\_ps\\_t prescaler](#)
- uint16\_t [finalValue](#)

#### Field Documentation

##### 16.131.2.4.1 ftm\_clock\_source\_t clockSelect

FTM clock source selection

Definition at line 131 of file timing\_pal.h.

##### 16.131.2.4.2 uint16\_t finalValue

The final value of FTM counter

Definition at line 133 of file timing\_pal.h.

##### 16.131.2.4.3 ftm\_clock\_ps\_t prescaler

Prescaler Selection

Definition at line 132 of file timing\_pal.h.

## 16.131.3 Enumeration Type Documentation

## 16.131.3.1 enum timer\_chan\_type\_t

Type options available for timer channel notification.

Implements : timer\_chan\_type\_t\_Class

## Enumerator

**TIMER\_CHAN\_TYPE\_CONTINUOUS** Timer channel creates continuous notification

**TIMER\_CHAN\_TYPE\_ONESHOT** Timer channel creates one-shot notification

Definition at line 65 of file timing\_pal.h.

## 16.131.3.2 enum timer\_resolution\_type\_t

Type options available for tick resolution.

Implements : timer\_resolution\_type\_t\_Class

## Enumerator

**TIMER\_RESOLUTION\_TYPE\_NANOSECOND** Tick resolution is nanosecond

**TIMER\_RESOLUTION\_TYPE\_MICROSECOND** Tick resolution is microsecond

**TIMER\_RESOLUTION\_TYPE\_MILLISECOND** Tick resolution is millisecond

Definition at line 53 of file timing\_pal.h.

## 16.131.4 Function Documentation

## 16.131.4.1 void TIMING\_Deinit ( const timing\_instance\_t \*const instance )

De-initialize a timer instance.

This function de-initializes timer instance. In order to use the timer instance again, TIMING\_Init must be called.

## Parameters

in	instance	The pointer to timer instance number structure
----	----------	--

Definition at line 604 of file timing\_pal.c.

## 16.131.4.2 void TIMING\_DisableNotification ( const timing\_instance\_t \*const instance, const uint8\_t channel )

Disable channel notifications.

This function disables channel notification.

## Parameters

in	instance	The pointer to timer instance number structure
in	channel	The channel number

Definition at line 1390 of file timing\_pal.c.

## 16.131.4.3 void TIMING\_EnableNotification ( const timing\_instance\_t \*const instance, const uint8\_t channel )

Enable channel notifications.

This function enables channel notification.

**Parameters**

in	<i>instance</i>	The pointer to timer instance number structure
in	<i>channel</i>	The channel number

Definition at line 1292 of file timing\_pal.c.

**16.131.4.4** `uint32_t TIMING_GetElapsed ( const timing_instance_t *const instance, const uint8_t channel )`

Get elapsed ticks.

This function gets elapsed time of the current period by ticks. The elapsed time by nanosecond, microsecond or millisecond is the result of this function multiplies by the result of the TIMING\_GetResolution function. Note that: If the timer channel type is continuous, this function may not return value of the period at the moment period is timeout depending on timer frequency, optimizations, etc. The behavior occurs if the execution time of the function is significant relative to timer tick duration. If the timer channel type is one-shot, this function can be used to check whether the current period is timeout, in this case if the returned value is bigger or equal than the period, the current period is timeout or overflowed.

**Parameters**

in	<i>instance</i>	The pointer to timer instance number structure
in	<i>channel</i>	The channel number

**Returns**

Number of ticks elapsed of the current period

Definition at line 956 of file timing\_pal.c.

**16.131.4.5** `status_t TIMING_GetMaxPeriod ( const timing_instance_t *const instance, const timer_resolution_type_t type, uint64_t *const maxPeriod )`

Get max period in engineering units.

This function gets max period in engineering units.

**Parameters**

in	<i>instance</i>	The pointer to timer instance number structure
in	<i>type</i>	Resolution type
out	<i>maxPeriod</i>	The pointer to max period in engineering units

**Returns**

Operation status

- STATUS\_SUCCESS: Operation was successful
- STATUS\_ERROR : The timer frequency is not fit to resolution type

Definition at line 1627 of file timing\_pal.c.

**16.131.4.6** `uint32_t TIMING_GetRemaining ( const timing_instance_t *const instance, const uint8_t channel )`

Get remaining ticks.

This function gets remaining time of the current period by ticks. The remaining time by nanosecond, microsecond or millisecond is the result of this function multiplies by the result of the TIMING\_GetResolution function. Note that: If the timer channel type is continuous, this function may not return 0 at the moment period is timeout depending on timer frequency, optimizations, etc. The behavior occurs if the execution time of the function is significant relative to timer tick duration. If the timer channel type is one-shot, this function can be used to check whether the current period is timeout, in this case if the returned value is 0, the current period is timeout or overflowed.

**Parameters**

in	<i>instance</i>	The pointer to timer instance number structure
in	<i>channel</i>	The channel number

**Returns**

Number of ticks remaining of the current period

Definition at line 1127 of file timing\_pal.c.

**16.131.4.7** `status_t TIMING_GetResolution ( const timing_instance_t *const instance, const timer_resolution_type_t type, uint64_t *const resolution )`

Get tick resolution.

This function gets tick resolution in engineering units (nanosecond, microsecond or millisecond). The result of this function is used to calculate period, remaining time or elapsed time in engineering units.

**Parameters**

in	<i>instance</i>	The pointer to timer instance number structure
in	<i>type</i>	Resolution type
out	<i>resolution</i>	The pointer to resolution in engineering units

**Returns**

Operation status

- STATUS\_SUCCESS: Operation was successful
- STATUS\_ERROR : The timer frequency is not fit to resolution type

Definition at line 1481 of file timing\_pal.c.

**16.131.4.8** `status_t TIMING_Init ( const timing_instance_t *const instance, const timer_config_t *const config )`

Initialize the timer instance and timer channels with value from input configuration structure.

This function initializes clock source, prescaler of the timer instance(except LPIT, PIT), the final value of counter (only FTM). This function also setups notification type and callback function of timer channel. The timer instance number and its configuration structure shall be passed as arguments. Timer channels do not start counting by default after calling this function. The function TIMING\_StartChannel must be called to start the timer channel counting.

**Parameters**

in	<i>instance</i>	The pointer to timer instance number structure
in	<i>config</i>	The pointer to configuration structure

**Returns**

Operation status

- STATUS\_SUCCESS: Operation was successful
- STATUS\_ERROR : Operation was fail if the timer instance is out of range For example: Timing over LPIT but the instance is not LPIT instance(TIMING\_OVER\_LPIT0\_INSTANCE)
- STATUS\_ERROR : Operation was fail if the FTM instance has been initialized

Definition at line 526 of file timing\_pal.c.

**16.131.4.9** void TIMING\_InstallCallback ( const timing\_instance\_t \*const *instance*, const uint8\_t *channel*, const timer\_callback\_t *callback*, void \*const *callbackParam* )

Installs callback function for the timer channel.

This function installs new callback function and callback parameter for the timer channel event. This function allows changing the callback function and parameter while the timer channel is running. If the provided callback function parameter is NULL, it is equivalent to removing the callback.

#### Parameters

in	<i>instance</i>	The pointer to timer instance number structure
in	<i>callback</i>	The new callback function for timer channel
in	<i>callbackParam</i>	The new callback parameter pointer

Definition at line 1785 of file timing\_pal.c.

**16.131.4.10** void TIMING\_StartChannel ( const timing\_instance\_t \*const *instance*, const uint8\_t *channel*, const uint32\_t *periodTicks* )

Starts the timer channel counting.

This function starts channel counting with a new period in ticks. Note that:

- If the timer is PIT or LPIT, to abort the current timer channel period and start a timer channel period with a new value, the timer channel must be stopped and started again.
- If the timer is FTM, this function start channel by enable channel interrupt generation.
- LPTMR and FTM is 16 bit timer, so the input period must be smaller than 65535.
- LPTMR and FTM is 16 bit timer, so the input period must be smaller than 65535.

#### Parameters

in	<i>instance</i>	The pointer to timer instance number structure
in	<i>channel</i>	The channel number
in	<i>periodTicks</i>	The input period in ticks

Definition at line 678 of file timing\_pal.c.

**16.131.4.11** void TIMING\_StopChannel ( const timing\_instance\_t \*const *instance*, const uint8\_t *channel* )

Stop the timer channel counting.

This function stop channel counting. Note that if the timer is FTM, this function stop channel by disable channel interrupt generation.

#### Parameters

in	<i>instance</i>	The pointer to timer instance number structure
in	<i>channel</i>	The channel number

Definition at line 854 of file timing\_pal.c.



## 16.132 Transceiver interrupt enable configuration options (0x23)

### 16.132.1 Detailed Description

This section lists all the possible options of 0x23 register related to:

- CAN bus silence interrupt
- LIN 2 wake-up interrupt
- LIN 1 wake-up interrupt
- CAN failure interrupt
- CAN wake-up interrupt

See also

[UJA113X\\_SetTransceiverInterrupts\(\)](#)

#### CAN bus silence interrupt configuration options (0x23:CBSIE[4])

- #define [CBSIE\\_DISABLE](#) 0x00U  
*Disable CAN bus silence interrupt.*
- #define [CBSIE\\_ENABLE](#) 0x10U  
*Enable CAN bus silence interrupt.*

#### LIN 2 wake-up interrupt configuration options (0x23:LWI2E[3])

- #define [L2WIE\\_DISABLE](#) 0x00U  
*Disable LIN 2 wake-up interrupt.*
- #define [L2WIE\\_ENABLE](#) 0x08U  
*Enable LIN 2 wake-up interrupt.*

#### LIN 1 wake-up interrupt configuration options (0x23:LWI1E[2])

- #define [L1WIE\\_DISABLE](#) 0x00U  
*Disable LIN 1 wake-up interrupt.*
- #define [L1WIE\\_ENABLE](#) 0x04U  
*Enable LIN 1 wake-up interrupt.*

#### CAN failure interrupt configuration options (0x23:CFIE[1])

- #define [CFIE\\_DISABLE](#) 0x00U  
*Disable CAN failure interrupt.*
- #define [CFIE\\_ENABLE](#) 0x02U  
*Enable CAN failure interrupt.*

#### CAN wake-up interrupt configuration options (0x23:CWIE[0])

- #define [CWIE\\_DISABLE](#) 0x00U  
*Disable CAN wake-up interrupt.*
- #define [CWIE\\_ENABLE](#) 0x01U  
*Enable CAN wake-up interrupt.*

### 16.132.2 Macro Definition Documentation

#### 16.132.2.1 `#define CBSIE_DISABLE 0x00U`

Disable CAN bus silence interrupt.

Definition at line 718 of file `sbc_uja113x.h`.

#### 16.132.2.2 `#define CBSIE_ENABLE 0x10U`

Enable CAN bus silence interrupt.

Definition at line 719 of file `sbc_uja113x.h`.

#### 16.132.2.3 `#define CFIE_DISABLE 0x00U`

Disable CAN failure interrupt.

Definition at line 739 of file `sbc_uja113x.h`.

#### 16.132.2.4 `#define CFIE_ENABLE 0x02U`

Enable CAN failure interrupt.

Definition at line 740 of file `sbc_uja113x.h`.

#### 16.132.2.5 `#define CWIE_DISABLE 0x00U`

Disable CAN wake-up interrupt.

Definition at line 746 of file `sbc_uja113x.h`.

#### 16.132.2.6 `#define CWIE_ENABLE 0x01U`

Enable CAN wake-up interrupt.

Definition at line 747 of file `sbc_uja113x.h`.

#### 16.132.2.7 `#define L1WIE_DISABLE 0x00U`

Disable LIN 1 wake-up interrupt.

Definition at line 732 of file `sbc_uja113x.h`.

#### 16.132.2.8 `#define L1WIE_ENABLE 0x04U`

Enable LIN 1 wake-up interrupt.

Definition at line 733 of file `sbc_uja113x.h`.

#### 16.132.2.9 `#define L2WIE_DISABLE 0x00U`

Disable LIN 2 wake-up interrupt.

Definition at line 725 of file `sbc_uja113x.h`.

#### 16.132.2.10 `#define L2WIE_ENABLE 0x08U`

Enable LIN 2 wake-up interrupt.

Definition at line 726 of file `sbc_uja113x.h`.

## 16.133 Transport layer API

### 16.133.1 Detailed Description

Transport layer stands between the application layer and the core API layer.

This layer consists the implementation of data transportation which contains one or more LIN frames. It is situated between the application layer and the core API layer including LIN2.1 TL API and LIN TL J2602. This layer provides APIs for the transport protocol, node configuration and diagnostic services. For LIN 2.1, all components will be extended from LIN 2.0 specification. The node configuration for J2602 implements only some functions of LIN 2.0 specification.

#### Modules

- [Common Transport Layer API](#)

*Contains Transport Layer APIs that used for both protocols LIN 2.1 and J2602.*

- [J2602 Transport Layer specific API](#)

*Contains Transport Layer APIs that only used for J2602 protocol.*

## 16.134 UJA116xA SBC Driver

### 16.134.1 Detailed Description

#### Data Structures

- struct [sbc\\_wtdog\\_ctr\\_t](#)  
Watchdog control register structure. Watchdog configuration structure. [More...](#)
- struct [sbc\\_sbc\\_t](#)  
SBC configuration control register structure. Two operating modes have a major impact on the operation of the watchdog: Forced Normal mode and Software Development mode (Software Development mode is provided for test and development purposes only and is not a dedicated SBC operating mode; the UJA116xA can be in any functional operating mode with Software Development mode enabled). These modes are enabled and disabled via bits FNMC and SDMC respectively in the SBC configuration control register. Note that this register is located in the non-volatile memory area. The watchdog is disabled in Forced Normal mode (FNM). In Software Development mode (SDM), the watchdog can be disabled or activated for test and software debugging purposes. [More...](#)
- struct [sbc\\_start\\_up\\_t](#)  
Start-up control register structure. This structure contains settings of RSTN output reset pulse width and V2/VEXT start-up control. [More...](#)
- struct [sbc\\_regulator\\_t](#)  
Regulator control register structure. This structure set power distribution control, V2/VEXT configuration, set V1 reset threshold. [More...](#)
- struct [sbc\\_supply\\_evt\\_t](#)  
Supply event capture enable register structure. This structure enables or disables detection of V2/VEXT overvoltage, undervoltage and V1 undervoltage enable. [More...](#)
- struct [sbc\\_sys\\_evt\\_t](#)  
System event capture enable register structure. This structure enables or disables overtemperature warning, SPI failure enable. [More...](#)
- struct [sbc\\_can\\_ctr\\_t](#)  
CAN control register structure. This structure configure CAN peripheral behavior. [More...](#)
- struct [sbc\\_trans\\_evt\\_t](#)  
Transceiver event capture enable register structure. Can bus silence, Can failure and Can wake-up settings. [More...](#)
- struct [sbc\\_frame\\_t](#)  
Frame control register structure. The wake-up frame format, standard (11-bit) or extended (29-bit) identifier, is selected via bit IDE in the Frame control register. [More...](#)
- struct [sbc\\_can\\_conf\\_t](#)  
CAN configuration group structure. This structure configure CAN peripheral behavior. [More...](#)
- struct [sbc\\_wake\\_t](#)  
WAKE pin event capture enable register structure. Local wake-up is enabled via bits WPRE and WPFE in the WAKE pin event capture enable register. A wake-up event is triggered by a LOW-to-HIGH (if WPRE = 1) and/or a HIGH-to-LOW (if WPFE = 1) transition on the WAKE pin. This arrangement allows for maximum flexibility when designing a local wake-up circuit. In applications that do not use the local wake-up facility, local wake-up should be disabled and the WAKE pin connected to GND. [More...](#)
- struct [sbc\\_regulator\\_ctr\\_t](#)  
Regulator control register group. This structure is group of regulator settings. [More...](#)
- struct [sbc\\_int\\_config\\_t](#)  
Init configuration structure. This structure is used for initialization of sbc. [More...](#)
- struct [sbc\\_factories\\_conf\\_t](#)  
Factory configuration structure. It contains Start-up control register and SBC configuration control register. This is non-volatile memory with limited write access. The MTPNV cells can be reprogrammed a maximum of 200 times (Ncy(W)MTP; Bit NVMPs in the MTPNV status register indicates whether the non-volatile cells can be reprogrammed. This register also contains a write counter, WRCNTS, that is incremented each time the MTPNV cells are reprogrammed (up to a maximum value of 111111; there is no overflow; performing a factory reset also increments the counter). This counter is provided for information purposes only; reprogramming will not be rejected when it reaches its maximum value. Factory preset values are restored if the following conditions apply continuously for at least td(MTPNV) during battery power-up: pin RSTN is held LOW, CANH is pulled up to VBAT, CANL is pulled down to

*GND After the factory preset values have been restored, the SBC performs a system reset and enters Forced normal Mode. Since the CAN-bus is clamped dominant, pin RXDC is forced LOW. Pin RXD is forced HIGH during the factory preset restore process (td(MTPNV)). A falling edge on RXD caused by bit PO being set after power-on indicates that the factory preset process has been completed. Note that the write counter, WRCNTS, in the MTPNV status register is incremented every time the factory presets are restored. [More...](#)*

- struct [sbc\\_main\\_status\\_t](#)

*Main status register structure. The Main status register can be accessed to monitor the status of the overtemperature warning flag and to determine whether the UJA116xA has entered Normal mode after initial power-up. It also indicates the source of the most recent reset event. [More...](#)*

- struct [sbc\\_wtdog\\_status\\_t](#)

*Watchdog status register structure. Information on the status of the watchdog is available from the Watchdog status register. This register also indicates whether Forced Normal and Software Development modes are active. [More...](#)*

- struct [sbc\\_supply\\_status\\_t](#)

*Supply voltage status register structure. V2/VEXT and V1 undervoltage and overvoltage status. [More...](#)*

- struct [sbc\\_trans\\_stat\\_t](#)

*Transceiver status register structure. There are stored CAN transceiver statuses. [More...](#)*

- struct [sbc\\_gl\\_evnt\\_stat\\_t](#)

*Global event status register. The microcontroller can monitor events via the event status registers. An extra status register, the Global event status register, is provided to help speed up software polling routines. By polling the Global event status register, the microcontroller can quickly determine the type of event captured (system, supply, transceiver or WAKE pin) and then query the relevant event status register. [More...](#)*

- struct [sbc\\_sys\\_evnt\\_stat\\_t](#)

*System event status register. Wake-up and interrupt event diagnosis in the UJA116xA is intended to provide the microcontroller with information on the status of a range of features and functions. This information is stored in the event status registers and is signaled on pin RXD, if enabled. [More...](#)*

- struct [sbc\\_sup\\_evnt\\_stat\\_t](#)

*Supply event status register. [More...](#)*

- struct [sbc\\_trans\\_evnt\\_stat\\_t](#)

*Transceiver event status register. [More...](#)*

- struct [sbc\\_wake\\_evnt\\_stat\\_t](#)

*WAKE pin event status register. [More...](#)*

- struct [sbc\\_evn\\_capt\\_t](#)

*Event capture registers structure. This structure contains Global event status, System event status, Supply event status, Transceiver event status, WAKE pin event status. [More...](#)*

- struct [sbc\\_mtpnv\\_stat\\_t](#)

*MTPNV status register. The MTPNV cells can be reprogrammed a maximum of 200 times (Ncy(W)MTP). Bit N<sub>VMPS</sub> in the MTPNV status register indicates whether the non-volatile cells can be reprogrammed. This register also contains a write counter, WRCNTS, that is incremented each time the MTPNV cells are reprogrammed (up to a maximum value of 111111; there is no overflow; performing a factory reset also increments the counter). This counter is provided for information purposes only; reprogramming will not be rejected when it reaches its maximum value. [More...](#)*

- struct [sbc\\_status\\_group\\_t](#)

*Status group structure. All statuses of SBC are stored in this structure. [More...](#)*

## Macros

- #define [SBC\\_UJA\\_TIMEOUT](#) 1000U
- #define [SBC\\_UJA\\_COUNT\\_ID\\_REG](#) 4U
- #define [SBC\\_UJA\\_COUNT\\_MASK](#) 4U
- #define [SBC\\_UJA\\_COUNT\\_DMASK](#) 8U

## Typedefs

- typedef uint8\_t [sbc\\_fail\\_safe\\_rcc\\_t](#)  
*Fail-safe control register, reset counter control (0x02). incremented every time the SBC enters Reset mode while FNMC = 0; RCC overflows from 11 to 00; default at power-on is 00.*
- typedef uint8\_t [sbc\\_identifier\\_t](#)  
*ID registers, identifier format (0x27 to 0x2A). A valid WUF identifier is defined and stored in the ID registers. An ID mask can be defined to allow a group of identifiers to be recognized as valid by an individual node.*
- typedef uint8\_t [sbc\\_identif\\_mask\\_t](#)  
*ID mask registers (0x2B to 0x2E). The identifier mask is defined in the ID mask registers, where a 1 means dont care.*
- typedef uint8\_t [sbc\\_frame\\_ctr\\_dlc\\_t](#)  
*Frame control register, number of data bytes expected in a CAN frame (0x2F).*
- typedef uint8\_t [sbc\\_data\\_mask\\_t](#)  
*Data mask registers. The data field indicates the nodes to be woken up. Within the data field, groups of nodes can be predefined and associated with bits in a data mask. By comparing the incoming data field with the data mask, multiple groups of nodes can be woken up simultaneously with a single wake-up message.*
- typedef uint8\_t [sbc\\_mtpnv\\_stat\\_wrnts\\_t](#)  
*MTPNV status register, write counter status (0x70). 6-bits - contains the number of times the MTPNV cells were reprogrammed.*

## Enumerations

- enum [sbc\\_register\\_t](#) {  
[SBC\\_UJA\\_WTDOG\\_CTR](#) = 0x00U, [SBC\\_UJA\\_MODE](#) = 0x01U, [SBC\\_UJA\\_FAIL\\_SAFE](#) = 0x02U, [SBC\\_UJA\\_MAIN](#) = 0x03U,  
[SBC\\_UJA\\_SYSTEM\\_EVNT](#) = 0x04U, [SBC\\_UJA\\_WTDOG\\_STAT](#) = 0x05U, [SBC\\_UJA\\_MEMORY\\_0](#) = 0x06U, [SBC\\_UJA\\_MEMORY\\_1](#) = 0x07U,  
[SBC\\_UJA\\_MEMORY\\_2](#) = 0x08U, [SBC\\_UJA\\_MEMORY\\_3](#) = 0x09U, [SBC\\_UJA\\_LOCK](#) = 0x0AU, [SBC\\_UJA\\_REGULATOR](#) = 0x0BU,  
[SBC\\_UJA\\_SUPPLY\\_STAT](#) = 0x0CU, [SBC\\_UJA\\_SUPPLY\\_EVNT](#) = 0x0DU, [SBC\\_UJA\\_CAN](#) = 0x0EU, [SBC\\_UJA\\_TRANS\\_STAT](#) = 0x0FU,  
[SBC\\_UJA\\_TRANS\\_EVNT](#) = 0x10U, [SBC\\_UJA\\_DAT\\_RATE](#) = 0x11U, [SBC\\_UJA\\_IDENTIF\\_0](#) = 0x12U, [SBC\\_UJA\\_IDENTIF\\_1](#) = 0x13U,  
[SBC\\_UJA\\_IDENTIF\\_2](#) = 0x14U, [SBC\\_UJA\\_IDENTIF\\_3](#) = 0x15U, [SBC\\_UJA\\_MASK\\_0](#) = 0x16U, [SBC\\_UJA\\_MASK\\_1](#) = 0x17U,  
[SBC\\_UJA\\_MASK\\_2](#) = 0x18U, [SBC\\_UJA\\_MASK\\_3](#) = 0x19U, [SBC\\_UJA\\_FRAME\\_CTR](#) = 0x1AU, [SBC\\_UJA\\_DAT\\_MASK\\_0](#) = 0x1BU,  
[SBC\\_UJA\\_DAT\\_MASK\\_1](#) = 0x1CU, [SBC\\_UJA\\_DAT\\_MASK\\_2](#) = 0x1DU, [SBC\\_UJA\\_DAT\\_MASK\\_3](#) = 0x1EU, [SBC\\_UJA\\_DAT\\_MASK\\_4](#) = 0x1FU,  
[SBC\\_UJA\\_DAT\\_MASK\\_5](#) = 0x20U, [SBC\\_UJA\\_DAT\\_MASK\\_6](#) = 0x21U, [SBC\\_UJA\\_DAT\\_MASK\\_7](#) = 0x22U, [SBC\\_UJA\\_WAKE\\_STAT](#) = 0x23U,  
[SBC\\_UJA\\_WAKE\\_EN](#) = 0x24U, [SBC\\_UJA\\_GL\\_EVNT\\_STAT](#) = 0x25U, [SBC\\_UJA\\_SYS\\_EVNT\\_STAT](#) = 0x26U, [SBC\\_UJA\\_SUP\\_EVNT\\_STAT](#) = 0x27U,  
[SBC\\_UJA\\_TRANS\\_EVNT\\_STAT](#) = 0x28U, [SBC\\_UJA\\_WAKE\\_EVNT\\_STAT](#) = 0x29U, [SBC\\_UJA\\_MTPNV\\_STAT](#) = 0x2AU, [SBC\\_UJA\\_START\\_UP](#) = 0x2BU,  
[SBC\\_UJA\\_SBC](#) = 0x2CU, [SBC\\_UJA\\_MTPNV\\_CRC](#) = 0x2DU, [SBC\\_UJA\\_IDENTIF](#) = 0x2EU }  
*Register map.*
- enum [sbc\\_wtdog\\_ctr\\_wmc\\_t](#) { [SBC\\_UJA\\_WTDOG\\_CTR\\_WMC\\_AUTO](#) = [SBC\\_UJA\\_WTDOG\\_CTR\\_WMC\\_F\(1U\)](#), [SBC\\_UJA\\_WTDOG\\_CTR\\_WMC\\_TIME](#) = [SBC\\_UJA\\_WTDOG\\_CTR\\_WMC\\_F\(2U\)](#), [SBC\\_UJA\\_WTDOG\\_CTR\\_WMC\\_WIND](#) = [SBC\\_UJA\\_WTDOG\\_CTR\\_WMC\\_F\(4U\)](#) }  
*Watchdog control register, watchdog mode control (0x00). The UJA116xA contains a watchdog that supports three operating modes: Window, Timeout and Autonomous. In Window mode (available only in SBC Normal mode), a watchdog trigger event within a defined watchdog window triggers and resets the watchdog timer. In Timeout mode, the watchdog runs continuously and can be triggered and reset at any time within the watchdog period by a watchdog trigger. Watchdog time-out mode can also be used for cyclic wake-up of the microcontroller. In Autonomous mode, the watchdog can be off or autonomously in Timeout mode, depending on the selected SBC mode. The watchdog mode*

is selected via bits WMC in the Watchdog control register. The SBC must be in Standby mode when the watchdog mode is changed.

- enum `sbc_wtdog_ctr_nwp_t` {  
`SBC_UJA_WTD OG_CTR_NWP_8` = 0x08U, `SBC_UJA_WTD OG_CTR_NWP_16` = 0x01U, `SBC_UJA_WTD OG_CTR_NWP_32` = 0x02U, `SBC_UJA_WTD OG_CTR_NWP_64` = 0x0BU,  
`SBC_UJA_WTD OG_CTR_NWP_128` = 0x04U, `SBC_UJA_WTD OG_CTR_NWP_256` = 0x0DU, `SBC_UJA_WTD OG_CTR_NWP_1024` = 0x0EU, `SBC_UJA_WTD OG_CTR_NWP_4096` = 0x07U }

Watchdog control register, nominal watchdog period (0x00). Eight watchdog periods are supported, from 8 ms to 4096 ms. The watchdog period is programmed via bits NWP. The selected period is valid for both Window and Timeout modes. The default watchdog period is 128 ms. A watchdog trigger event resets the watchdog timer. A watchdog trigger event is any valid write access to the Watchdog control register. If the watchdog mode or the watchdog period have changed as a result of the write access, the new values are immediately valid.

- enum `sbc_mode_mc_t` { `SBC_UJA_MODE_MC_SLEEP` = 0x01U, `SBC_UJA_MODE_MC_STANDBY` = 0x04U, `SBC_UJA_MODE_MC_NORMAL` = 0x07U }

Mode control register, mode control (0x01)

- enum `sbc_fail_safe_lhc_t` { `SBC_UJA_FAIL_SAFE_LHC_FLOAT` = `SBC_UJA_FAIL_SAFE_LHC_F(0U)`, `SBC_UJA_FAIL_SAFE_LHC_LOW` = `SBC_UJA_FAIL_SAFE_LHC_F(1U)` }

Fail-safe control register, LIMP home control (0x02). The dedicated LIMP pin can be used to enable so called limp home hardware in the event of a serious ECU failure. Detectable failure conditions include SBC overtemperature events, loss of watchdog service, short-circuits on pins RSTN or V1 and user-initiated or external reset events. The LIMP pin is a battery-robust, active-LOW, open-drain output. The LIMP pin can also be forced LOW by setting bit LHC in the Fail-safe control register.

- enum `sbc_main_otws_t` { `SBC_UJA_MAIN_OTWS_BELOW` = `SBC_UJA_MAIN_OTWS_F(0U)`, `SBC_UJA_MAIN_OTWS_ABOVE` = `SBC_UJA_MAIN_OTWS_F(1U)` }

Main status register, Overtemperature warning status (0x03).

- enum `sbc_main_nms_t` { `SBC_UJA_MAIN_NMS_NORMAL` = `SBC_UJA_MAIN_NMS_F(0U)`, `SBC_UJA_MAIN_NMS_PWR_UP` = `SBC_UJA_MAIN_NMS_F(1U)` }

Main status register, normal mode status (0x03).

- enum `sbc_main_rss_t` {  
`SBC_UJA_MAIN_RSS_OFF_MODE` = 0x00U, `SBC_UJA_MAIN_RSS_CAN_WAKEUP` = 0x01U, `SBC_UJA_MAIN_RSS_SLP_WAKEUP` = 0x04U, `SBC_UJA_MAIN_RSS_OVF_SLP` = 0x0CU,  
`SBC_UJA_MAIN_RSS_DIAG_WAKEUP` = 0x0DU, `SBC_UJA_MAIN_RSS_WATCH_TRIG` = 0x0EU, `SBC_UJA_MAIN_RSS_WATCH_OVF` = 0x0FU, `SBC_UJA_MAIN_RSS_ILLEG_WATCH` = 0x10U,  
`SBC_UJA_MAIN_RSS_RSTN_PULDW` = 0x11U, `SBC_UJA_MAIN_RSS_LFT_OVERTM` = 0x12U, `SBC_UJA_MAIN_RSS_V1_UNDERV` = 0x13U, `SBC_UJA_MAIN_RSS_ILLEG_SLP` = 0x14U,  
`SBC_UJA_MAIN_RSS_WAKE_SLP` = 0x16U }

Main status register, Reset source status (0x03).

- enum `sbc_sys_evnt_otwe_t` { `SBC_UJA_SYS_EVNT_OTWE_DIS` = `SBC_UJA_SYS_EVNT_OTWE_F(0U)`, `SBC_UJA_SYS_EVNT_OTWE_EN` = `SBC_UJA_SYS_EVNT_OTWE_F(1U)` }

System event capture enable, overtemperature warning enable (0x04).

- enum `sbc_sys_evnt_spipe_t` { `SBC_UJA_SYS_EVNT_SPIFE_DIS` = `SBC_UJA_SYS_EVNT_SPIFE_F(0U)`, `SBC_UJA_SYS_EVNT_SPIFE_EN` = `SBC_UJA_SYS_EVNT_SPIFE_F(1U)` }

System event capture enable, SPI failure enable (0x04).

- enum `sbc_wtdog_stat_fnms_t` { `SBC_UJA_WTD OG_STAT_FNMS_N_NORMAL` = `SBC_UJA_WTD OG_STAT_FNMS_F(0U)`, `SBC_UJA_WTD OG_STAT_FNMS_NORMAL` = `SBC_UJA_WTD OG_STAT_FNMS_F(1U)` }

Watchdog status register, forced Normal mode status (0x05).

- enum `sbc_wtdog_stat_sdms_t` { `SBC_UJA_WTD OG_STAT_SDMS_N_NORMAL` = `SBC_UJA_WTD OG_STAT_SDMS_F(0U)`, `SBC_UJA_WTD OG_STAT_SDMS_NORMAL` = `SBC_UJA_WTD OG_STAT_SDMS_F(1U)` }

Watchdog status register, Software Development mode status (0x05).

- enum `sbc_wtdog_stat_wds_t` { `SBC_UJA_WTD OG_STAT_WDS_OFF` = `SBC_UJA_WTD OG_STAT_WDS_F(0U)`, `SBC_UJA_WTD OG_STAT_WDS_FIH` = `SBC_UJA_WTD OG_STAT_WDS_F(1U)`, `SBC_UJA_WTD OG_STAT_WDS_SEH` = `SBC_UJA_WTD OG_STAT_WDS_F(2U)` }

Watchdog status register, watchdog status (0x05).

- enum `sbc_lock_t` {  
`LK0C` = `SBC_UJA_LOCK_LK0C_MASK`, `LK1C` = `SBC_UJA_LOCK_LK1C_MASK`, `LK2C` = `SBC_UJA_LOCK_LK2C_MASK`, `LK3C` = `SBC_UJA_LOCK_LK3C_MASK`,  
`LK4C` = `SBC_UJA_LOCK_LK4C_MASK`, `LK5C` = `SBC_UJA_LOCK_LK5C_MASK`, `LK6C` = `SBC_UJA_LOCK_LK6C_MASK`, `LKAC` = `SBC_UJA_LOCK_LKNC_MASK` }  
*Lock control(0x0A). Sections of the register address area can be write-protected to protect against unintended modifications. This facility only protects locked bits from being modified via the SPI and will not prevent the UJA116xA updating status registers etc.*
- enum `sbc_regulator_pdc_t` { `SBC_UJA_REGULATOR_PDC_HV` = `SBC_UJA_REGULATOR_PDC_F(0U)`,  
`SBC_UJA_REGULATOR_PDC_LV` = `SBC_UJA_REGULATOR_PDC_F(1U)` }  
*Regulator control register, power distribution control (0x10). PDC is not available on UJA1168 device variants, use any of these two values, the value written to the device will be ignored.*
- enum `sbc_regulator_v2c_t` { `SBC_UJA_REGULATOR_V2C_OFF` = `SBC_UJA_REGULATOR_V2C_F(0U)`,  
`SBC_UJA_REGULATOR_V2C_N` = `SBC_UJA_REGULATOR_V2C_F(1U)`, `SBC_UJA_REGULATOR_V2C_N_S_S_R` = `SBC_UJA_REGULATOR_V2C_F(2U)`, `SBC_UJA_REGULATOR_V2C_N_S_S_R` = `SBC_UJA_REGULATOR_V2C_F(3U)` }  
*Regulator control register, V2/VEXT configuration (0x10).*
- enum `sbc_regulator_v1rtc_t` { `SBC_UJA_REGULATOR_V1RTC_90` = `SBC_UJA_REGULATOR_V1RTC_F(0U)`, `SBC_UJA_REGULATOR_V1RTC_80` = `SBC_UJA_REGULATOR_V1RTC_F(1U)`, `SBC_UJA_REGULATOR_V1RTC_70` = `SBC_UJA_REGULATOR_V1RTC_F(2U)`, `SBC_UJA_REGULATOR_V1RTC_60` = `SBC_UJA_REGULATOR_V1RTC_F(3U)` }  
*Regulator control register, set V1 reset threshold (0x10).*
- enum `sbc_supply_stat_v2s_t` { `SBC_UJA_SUPPLY_STAT_V2S_VOK` = `SBC_UJA_SUPPLY_STAT_V2S_F(0U)`, `SBC_UJA_SUPPLY_STAT_V2S_VBE` = `SBC_UJA_SUPPLY_STAT_V2S_F(1U)`, `SBC_UJA_SUPPLY_STAT_V2S_VAB` = `SBC_UJA_SUPPLY_STAT_V2S_F(2U)`, `SBC_UJA_SUPPLY_STAT_V2S_DIS` = `SBC_UJA_SUPPLY_STAT_V2S_F(3U)` }  
*Supply voltage status register, V2/VEXT status (0x1B).*
- enum `sbc_supply_stat_v1s_t` { `SBC_UJA_SUPPLY_STAT_V1S_VAB` = `SBC_UJA_SUPPLY_STAT_V1S_F(0U)`, `SBC_UJA_SUPPLY_STAT_V1S_VBE` = `SBC_UJA_SUPPLY_STAT_V1S_F(1U)` }  
*Supply voltage status register, V1 status (0x1B).*
- enum `sbc_supply_evnt_v2oe_t` { `SBC_UJA_SUPPLY_EVNT_V2OE_DIS` = `SBC_UJA_SUPPLY_EVNT_V2OE_F(0U)`, `SBC_UJA_SUPPLY_EVNT_V2OE_EN` = `SBC_UJA_SUPPLY_EVNT_V2OE_F(1U)` }  
*Supply event capture enable register, V2/VEXT overvoltage enable (0x1C).*
- enum `sbc_supply_evnt_v2ue_t` { `SBC_UJA_SUPPLY_EVNT_V2UE_DIS` = `SBC_UJA_SUPPLY_EVNT_V2UE_F(0U)`, `SBC_UJA_SUPPLY_EVNT_V2UE_EN` = `SBC_UJA_SUPPLY_EVNT_V2UE_F(1U)` }  
*Supply event capture enable register, V2/VEXT undervoltage enable (0x1C).*
- enum `sbc_supply_evnt_v1ue_t` { `SBC_UJA_SUPPLY_EVNT_V1UE_DIS` = `SBC_UJA_SUPPLY_EVNT_V1UE_F(0U)`, `SBC_UJA_SUPPLY_EVNT_V1UE_EN` = `SBC_UJA_SUPPLY_EVNT_V1UE_F(1U)` }  
*Supply event capture enable register, V1 undervoltage enable (0x1C).*
- enum `sbc_can_cfdc_t` { `SBC_UJA_CAN_CFDC_DIS` = `SBC_UJA_CAN_CFDC_F(0U)`, `SBC_UJA_CAN_CFDC_EN` = `SBC_UJA_CAN_CFDC_F(1U)` }  
*CAN control register, CAN FD control (0x20).*
- enum `sbc_can_pncok_t` { `SBC_UJA_CAN_PNCOK_DIS` = `SBC_UJA_CAN_PNCOK_F(0U)`, `SBC_UJA_CAN_PNCOK_EN` = `SBC_UJA_CAN_PNCOK_F(1U)` }  
*CAN control register, CAN partial networking configuration OK (0x20).*
- enum `sbc_can_cpnc_t` { `SBC_UJA_CAN_CPNC_DIS` = `SBC_UJA_CAN_CPNC_F(0U)`, `SBC_UJA_CAN_CPNC_EN` = `SBC_UJA_CAN_CPNC_F(1U)` }  
*CAN control register, CAN partial networking control (0x20).*
- enum `sbc_can_cmc_t` { `SBC_UJA_CAN_CMC_OFMODE` = `SBC_UJA_CAN_CMC_F(0U)`, `SBC_UJA_CAN_CMC_ACMODE_DA` = `SBC_UJA_CAN_CMC_F(1U)`, `SBC_UJA_CAN_CMC_ACMODE_DD` = `SBC_UJA_CAN_CMC_F(2U)`, `SBC_UJA_CAN_CMC_LISTEN` = `SBC_UJA_CAN_CMC_F(3U)` }  
*CAN control register, CAN mode control (0x20).*
- enum `sbc_trans_stat_cts_t` { `SBC_UJA_TRANS_STAT_CTS_INACT` = `SBC_UJA_TRANS_STAT_CTS_F(0U)`, `SBC_UJA_TRANS_STAT_CTS_ACT` = `SBC_UJA_TRANS_STAT_CTS_F(1U)` }



*Transceiver status register, CAN transceiver status (0x22).*

- enum `sbc_trans_stat_cpnrerr_t` { `SBC_UJA_TRANS_STAT_CPNRERR_NO_DET` = `SBC_UJA_TRANS_STAT_CPNRERR_F(0U)`, `SBC_UJA_TRANS_STAT_CPNRERR_DET` = `SBC_UJA_TRANS_STAT_CPNRERR_F(1U)` }

*Transceiver status register, CAN partial networking error (0x22).*

- enum `sbc_trans_stat_cpns_t` { `SBC_UJA_TRANS_STAT_CPNS_ERR` = `SBC_UJA_TRANS_STAT_CPNS_F(0U)`, `SBC_UJA_TRANS_STAT_CPNS_OK` = `SBC_UJA_TRANS_STAT_CPNS_F(1U)` }

*Transceiver status register, CAN partial networking status (0x22).*

- enum `sbc_trans_stat_coscs_t` { `SBC_UJA_TRANS_STAT_COSCS_NRUN` = `SBC_UJA_TRANS_STAT_COSCS_F(0U)`, `SBC_UJA_TRANS_STAT_COSCS_RUN` = `SBC_UJA_TRANS_STAT_COSCS_F(1U)` }

*Transceiver status register, CAN oscillator status (0x22).*

- enum `sbc_trans_stat_cbss_t` { `SBC_UJA_TRANS_STAT_CBSS_ACT` = `SBC_UJA_TRANS_STAT_CBSS_F(0U)`, `SBC_UJA_TRANS_STAT_CBSS_INACT` = `SBC_UJA_TRANS_STAT_CBSS_F(1U)` }

*Transceiver status register, CAN-bus silence status (0x22).*

- enum `sbc_trans_stat_vcs_t` { `SBC_UJA_TRANS_STAT_VCS_AB` = `SBC_UJA_TRANS_STAT_VCS_F(0U)`, `SBC_UJA_TRANS_STAT_VCS_BE` = `SBC_UJA_TRANS_STAT_VCS_F(1U)` }

*Transceiver status register, VCAN status (0x22).*

- enum `sbc_trans_stat_cfs_t` { `SBC_UJA_TRANS_STAT_CFS_NO_TXD` = `SBC_UJA_TRANS_STAT_CFS_F(0U)`, `SBC_UJA_TRANS_STAT_CFS_TXD` = `SBC_UJA_TRANS_STAT_CFS_F(1U)` }

*Transceiver status register, CAN failure status (0x22).*

- enum `sbc_trans_evnt_cbse_t` { `SBC_UJA_TRANS_EVNT_CBSE_DIS` = `SBC_UJA_TRANS_EVNT_CBSE_F(0U)`, `SBC_UJA_TRANS_EVNT_CBSE_EN` = `SBC_UJA_TRANS_EVNT_CBSE_F(1U)` }

*Transceiver event capture enable register, CAN-bus silence enable (0x23).*

- enum `sbc_trans_evnt_cfe_t` { `SBC_UJA_TRANS_EVNT_CFE_DIS` = `SBC_UJA_TRANS_EVNT_CFE_F(0U)`, `SBC_UJA_TRANS_EVNT_CFE_EN` = `SBC_UJA_TRANS_EVNT_CFE_F(1U)` }

*Transceiver event capture enable register, CAN failure enable (0x23).*

- enum `sbc_trans_evnt_cwe_t` { `SBC_UJA_TRANS_EVNT_CWE_DIS` = `SBC_UJA_TRANS_EVNT_CWE_F(0U)`, `SBC_UJA_TRANS_EVNT_CWE_EN` = `SBC_UJA_TRANS_EVNT_CWE_F(1U)` }

*Transceiver event capture enable register, CAN wake-up enable (0x23).*

- enum `sbc_dat_rate_t` { `SBC_UJA_DAT_RATE_CDR_50KB` = `SBC_UJA_DAT_RATE_CDR_F(0U)`, `SBC_UJA_DAT_RATE_CDR_100KB` = `SBC_UJA_DAT_RATE_CDR_F(1U)`, `SBC_UJA_DAT_RATE_CDR_125KB` = `SBC_UJA_DAT_RATE_CDR_F(2U)`, `SBC_UJA_DAT_RATE_CDR_250KB` = `SBC_UJA_DAT_RATE_CDR_F(3U)`, `SBC_UJA_DAT_RATE_CDR_500KB` = `SBC_UJA_DAT_RATE_CDR_F(5U)`, `SBC_UJA_DAT_RATE_CDR_1000KB` = `SBC_UJA_DAT_RATE_CDR_F(7U)` }

*Data rate register, CAN data rate selection (0x26). CAN partial networking configuration registers. Dedicated registers are provided for configuring CAN partial networking.*

- enum `sbc_frame_ctr_ide_t` { `SBC_UJA_FRAME_CTR_IDE_11B` = `SBC_UJA_FRAME_CTR_IDE_F(0U)`, `SBC_UJA_FRAME_CTR_IDE_29B` = `SBC_UJA_FRAME_CTR_IDE_F(1U)` }

*Frame control register, identifier format (0x2F). The wake-up frame format, standard (11-bit) or extended (29-bit) identifier, is selected via bit IDE in the Frame control register.*

- enum `sbc_frame_ctr_pndm_t` { `SBC_UJA_FRAME_CTR_PNDM_DCARE` = `SBC_UJA_FRAME_CTR_PNDM_F(0U)`, `SBC_UJA_FRAME_CTR_PNDM_EVAL` = `SBC_UJA_FRAME_CTR_PNDM_F(1U)` }

*Frame control register, partial networking data mask (0x2F).*

- enum `sbc_wake_stat_wpvs_t` { `SBC_UJA_WAKE_STAT_WPVS_BE` = `SBC_UJA_WAKE_STAT_WPVS_F(0U)`, `SBC_UJA_WAKE_STAT_WPVS_AB` = `SBC_UJA_WAKE_STAT_WPVS_F(1U)` }

*WAKE pin status register, WAKE pin status (0x4B).*

- enum `sbc_wake_en_wpre_t` { `SBC_UJA_WAKE_EN_WPRE_DIS` = `SBC_UJA_WAKE_EN_WPRE_F(0U)`, `SBC_UJA_WAKE_EN_WPRE_EN` = `SBC_UJA_WAKE_EN_WPRE_F(1U)` }

*WAKE pin event capture enable register, WAKE pin rising-edge enable (0x4C).*

- enum `sbc_wake_en_wpfe_t` { `SBC_UJA_WAKE_EN_WPFE_DIS` = `SBC_UJA_WAKE_EN_WPFE_F(0U)`, `SBC_UJA_WAKE_EN_WPFE_EN` = `SBC_UJA_WAKE_EN_WPFE_F(1U)` }

*WAKE pin event capture enable register, WAKE pin falling-edge enable (0x4C).*

- enum `sbc_gl_evnt_stat_wpe_t` { `SBC_UJA_GL_EVNT_STAT_WPE_NO` = `SBC_UJA_GL_EVNT_STAT_WPE_F(0U)`, `SBC_UJA_GL_EVNT_STAT_WPE` = `SBC_UJA_GL_EVNT_STAT_WPE_F(1U)` }  
*Global event status register, WAKE pin event (0x60).*
- enum `sbc_gl_evnt_stat_trxe_t` { `SBC_UJA_GL_EVNT_STAT_TRXE_NO` = `SBC_UJA_GL_EVNT_STAT_TRXE_F(0U)`, `SBC_UJA_GL_EVNT_STAT_TRXE` = `SBC_UJA_GL_EVNT_STAT_TRXE_F(1U)` }  
*Global event status register, transceiver event (0x60).*
- enum `sbc_gl_evnt_stat_supe_t` { `SBC_UJA_GL_EVNT_STAT_SUPE_NO` = `SBC_UJA_GL_EVNT_STAT_SUPE_F(0U)`, `SBC_UJA_GL_EVNT_STAT_SUPE` = `SBC_UJA_GL_EVNT_STAT_SUPE_F(1U)` }  
*Global event status register, supply event (0x60).*
- enum `sbc_gl_evnt_stat_syse_t` { `SBC_UJA_GL_EVNT_STAT_SYSE_NO` = `SBC_UJA_GL_EVNT_STAT_SYSE_F(0U)`, `SBC_UJA_GL_EVNT_STAT_SYSE` = `SBC_UJA_GL_EVNT_STAT_SYSE_F(1U)` }  
*Global event status register, system event (0x60).*
- enum `sbc_sys_evnt_stat_po_t` { `SBC_UJA_SYS_EVNT_STAT_PO_NO` = `SBC_UJA_SYS_EVNT_STAT_PO_F(0U)`, `SBC_UJA_SYS_EVNT_STAT_PO` = `SBC_UJA_SYS_EVNT_STAT_PO_F(1U)` }  
*System event status register, power-on (0x61).*
- enum `sbc_sys_evnt_stat_otw_t` { `SBC_UJA_SYS_EVNT_STAT_OTW_NO` = `SBC_UJA_SYS_EVNT_STAT_OTW_F(0U)`, `SBC_UJA_SYS_EVNT_STAT_OTW` = `SBC_UJA_SYS_EVNT_STAT_OTW_F(1U)` }  
*System event status register, overtemperature warning (0x61).*
- enum `sbc_sys_evnt_stat_spif_t` { `SBC_UJA_SYS_EVNT_STAT_SPIF_NO` = `SBC_UJA_SYS_EVNT_STAT_SPIF_F(0U)`, `SBC_UJA_SYS_EVNT_STAT_SPIF` = `SBC_UJA_SYS_EVNT_STAT_SPIF_F(1U)` }  
*System event status register, SPI failure (0x61).*
- enum `sbc_sys_evnt_stat_wdf_t` { `SBC_UJA_SYS_EVNT_STAT_WDF_NO` = `SBC_UJA_SYS_EVNT_STAT_WDF_F(0U)`, `SBC_UJA_SYS_EVNT_STAT_WDF` = `SBC_UJA_SYS_EVNT_STAT_WDF_F(1U)` }  
*System event status register, watchdog failure (0x61).*
- enum `sbc_sup_evnt_stat_v2o_t` { `SBC_UJA_SUP_EVNT_STAT_V2O_NO` = `SBC_UJA_SUP_EVNT_STAT_V2O_F(0U)`, `SBC_UJA_SUP_EVNT_STAT_V2O` = `SBC_UJA_SUP_EVNT_STAT_V2O_F(1U)` }  
*Supply event status register, V2/VEXT overvoltage (0x62).*
- enum `sbc_sup_evnt_stat_v2u_t` { `SBC_UJA_SUP_EVNT_STAT_V2U_NO` = `SBC_UJA_SUP_EVNT_STAT_V2U_F(0U)`, `SBC_UJA_SUP_EVNT_STAT_V2U` = `SBC_UJA_SUP_EVNT_STAT_V2U_F(1U)` }  
*Supply event status register, V2/VEXT undervoltage (0x62).*
- enum `sbc_sup_evnt_stat_v1u_t` { `SBC_UJA_SUP_EVNT_STAT_V1U_NO` = `SBC_UJA_SUP_EVNT_STAT_V1U_F(0U)`, `SBC_UJA_SUP_EVNT_STAT_V1U` = `SBC_UJA_SUP_EVNT_STAT_V1U_F(1U)` }  
*Supply event status register, V1 undervoltage (0x62).*
- enum `sbc_trans_evnt_stat_pnfde_t` { `SBC_UJA_TRANS_EVNT_STAT_PNFDE_NO` = `SBC_UJA_TRANS_EVNT_STAT_PNFDE_F(0U)`, `SBC_UJA_TRANS_EVNT_STAT_PNFDE` = `SBC_UJA_TRANS_EVNT_STAT_PNFDE_F(1U)` }  
*Transceiver event status register, partial networking frame detection error (0x63).*
- enum `sbc_trans_evnt_stat_cbs_t` { `SBC_UJA_TRANS_EVNT_STAT_CBS_NO` = `SBC_UJA_TRANS_EVNT_STAT_CBS_F(0U)`, `SBC_UJA_TRANS_EVNT_STAT_CBS` = `SBC_UJA_TRANS_EVNT_STAT_CBS_F(1U)` }  
*Transceiver event status register, CAN-bus status (0x63).*
- enum `sbc_trans_evnt_stat_cf_t` { `SBC_UJA_TRANS_EVNT_STAT_CF_NO` = `SBC_UJA_TRANS_EVNT_STAT_CF_F(0U)`, `SBC_UJA_TRANS_EVNT_STAT_CF` = `SBC_UJA_TRANS_EVNT_STAT_CF_F(1U)` }  
*Transceiver event status register, CAN failure (0x63).*
- enum `sbc_trans_evnt_stat_cw_t` { `SBC_UJA_TRANS_EVNT_STAT_CW_NO` = `SBC_UJA_TRANS_EVNT_STAT_CW_F(0U)`, `SBC_UJA_TRANS_EVNT_STAT_CW` = `SBC_UJA_TRANS_EVNT_STAT_CW_F(1U)` }  
*Transceiver event status register, CAN wake-up (0x63).*
- enum `sbc_wake_evnt_stat_wpr_t` { `SBC_UJA_WAKE_EVNT_STAT_WPR_NO` = `SBC_UJA_WAKE_EVNT_STAT_WPR_F(0U)`, `SBC_UJA_WAKE_EVNT_STAT_WPR` = `SBC_UJA_WAKE_EVNT_STAT_WPR_F(1U)` }  
*WAKE pin event status register, WAKE pin rising edge (0x64).*

- enum `sbc_wake_evnt_stat_wpf_t` { `SBC_UJA_WAKE_EVNT_STAT_WPF_NO` = `SBC_UJA_WAKE_EVNT_STAT_WPF_F(0U)`, `SBC_UJA_WAKE_EVNT_STAT_WPF` = `SBC_UJA_WAKE_EVNT_STAT_WPF_F(1U)` }  
*WAKE pin event status register, WAKE pin falling edge (0x64).*
- enum `sbc_mtpnv_stat_eccs_t` { `SBC_UJA_MTPNV_STAT_ECCS_NO` = `SBC_UJA_MTPNV_STAT_ECCS_F(0U)`, `SBC_UJA_MTPNV_STAT_ECCS` = `SBC_UJA_MTPNV_STAT_ECCS_F(1U)` }  
*MTPNV status register, error correction code status (0x70).*
- enum `sbc_mtpnv_stat_nvmps_t` { `SBC_UJA_MTPNV_STAT_NVMP_NO` = `SBC_UJA_MTPNV_STAT_NVMP_F(0U)`, `SBC_UJA_MTPNV_STAT_NVMP` = `SBC_UJA_MTPNV_STAT_NVMP_F(1U)` }  
*MTPNV status register, non-volatile memory programming status (0x70).*
- enum `sbc_start_up_rlc_t` { `SBC_UJA_START_UP_RLC_20_25p0` = `SBC_UJA_START_UP_RLC_F(0U)`, `SBC_UJA_START_UP_RLC_10_12p5` = `SBC_UJA_START_UP_RLC_F(1U)`, `SBC_UJA_START_UP_RLC_03p6_05` = `SBC_UJA_START_UP_RLC_F(2U)`, `SBC_UJA_START_UP_RLC_01_01p5` = `SBC_UJA_START_UP_RLC_F(3U)` }  
*Start-up control register, RSTN output reset pulse width macros (0x73).*
- enum `sbc_start_up_v2suc_t` { `SBC_UJA_START_UP_V2SUC_00` = `SBC_UJA_START_UP_V2SUC_F(0U)`, `SBC_UJA_START_UP_V2SUC_11` = `SBC_UJA_START_UP_V2SUC_F(1U)` }  
*Start-up control register, V2/VEXT start-up control (0x73).*
- enum `sbc_sbc_v1rtsuc_t` { `SBC_UJA_SBC_V1RTSUC_90` = `SBC_UJA_SBC_V1RTSUC_F(0U)`, `SBC_UJA_SBC_V1RTSUC_80` = `SBC_UJA_SBC_V1RTSUC_F(1U)`, `SBC_UJA_SBC_V1RTSUC_70` = `SBC_UJA_SBC_V1RTSUC_F(2U)`, `SBC_UJA_SBC_V1RTSUC_60` = `SBC_UJA_SBC_V1RTSUC_F(3U)` }  
*SBC configuration control register, V1 undervoltage threshold (defined by bit V1RTC) at start-up (0x74).*
- enum `sbc_sbc_fnmc_t` { `SBC_UJA_SBC_FNMC_DIS` = `SBC_UJA_SBC_FNMC_F(0U)`, `SBC_UJA_SBC_FNMC_EN` = `SBC_UJA_SBC_FNMC_F(1U)` }  
*SBC configuration control register, Forced Normal mode control (0x74).*
- enum `sbc_sbc_sdmc_t` { `SBC_UJA_SBC_SDMC_DIS` = `SBC_UJA_SBC_SDMC_F(0U)`, `SBC_UJA_SBC_SDMC_EN` = `SBC_UJA_SBC_SDMC_F(1U)` }  
*SBC configuration control register, Software Development mode control (0x74).*
- enum `sbc_sbc_slpc_t` { `SBC_UJA_SBC_SLPC_AC` = `SBC_UJA_SBC_SLPC_F(0U)`, `SBC_UJA_SBC_SLPC_IG` = `SBC_UJA_SBC_SLPC_F(1U)` }  
*SBC configuration control register, Sleep control (0x74).*

## 16.134.2 Data Structure Documentation

### 16.134.2.1 struct `sbc_wtdog_ctr_t`

Watchdog control register structure. Watchdog configuration structure.

Implements : `sbc_wtdog_ctr_t_Class`

Definition at line 1086 of file `sbc_uja116x_driver.h`.

#### Data Fields

- `sbc_wtdog_ctr_wmc_t` modeControl
- `sbc_wtdog_ctr_nwp_t` nominalPeriod

#### Field Documentation

##### 16.134.2.1.1 `sbc_wtdog_ctr_wmc_t` modeControl

Watchdog mode control.

Definition at line 1087 of file `sbc_uja116x_driver.h`.

#### 16.134.2.1.2 **sbc\_wtdog\_ctr\_nwp\_t** nominalPeriod

Nominal watchdog period.

Definition at line 1088 of file sbc\_uja116x\_driver.h.

#### 16.134.2.2 **struct sbc\_sbc\_t**

SBC configuration control register structure. Two operating modes have a major impact on the operation of the watchdog: Forced Normal mode and Software Development mode (Software Development mode is provided for test and development purposes only and is not a dedicated SBC operating mode; the UJA116xA can be in any functional operating mode with Software Development mode enabled). These modes are enabled and disabled via bits FNMC and SDMC respectively in the SBC configuration control register. Note that this register is located in the non-volatile memory area. The watchdog is disabled in Forced Normal mode (FNM). In Software Development mode (SDM), the watchdog can be disabled or activated for test and software debugging purposes.

Implements : `sbc_sbc_t_Class`

Definition at line 1107 of file sbc\_uja116x\_driver.h.

#### Data Fields

- [sbc\\_sbc\\_v1rtsuc\\_t v1rtsuc](#)
- [sbc\\_sbc\\_fnmc\\_t fnmc](#)
- [sbc\\_sbc\\_sdmc\\_t sdmc](#)
- [sbc\\_sbc\\_slpc\\_t slpc](#)

#### Field Documentation

##### 16.134.2.2.1 **sbc\_sbc\_fnmc\_t** fnmc

Forced Normal mode control.

Definition at line 1110 of file sbc\_uja116x\_driver.h.

##### 16.134.2.2.2 **sbc\_sbc\_sdmc\_t** sdmc

Software Development mode control.

Definition at line 1111 of file sbc\_uja116x\_driver.h.

##### 16.134.2.2.3 **sbc\_sbc\_slpc\_t** slpc

Sleep control.

Definition at line 1113 of file sbc\_uja116x\_driver.h.

##### 16.134.2.2.4 **sbc\_sbc\_v1rtsuc\_t** v1rtsuc

V1 undervoltage threshold (defined by bit V1RTC) at start-up (0x74).

Definition at line 1108 of file sbc\_uja116x\_driver.h.

#### 16.134.2.3 **struct sbc\_start\_up\_t**

Start-up control register structure. This structure contains settings of RSTN output reset pulse width and V2/VEXT start-up control.

Implements : `sbc_start_up_t_Class`

Definition at line 1123 of file sbc\_uja116x\_driver.h.

#### Data Fields

- [sbc\\_start\\_up\\_rlc\\_t rlc](#)
- [sbc\\_start\\_up\\_v2suc\\_t v2suc](#)

## Field Documentation

16.134.2.3.1 `sbc_start_up_rlc_t rlc`

RSTN output reset pulse width macros.

Definition at line 1124 of file `sbc_uja116x_driver.h`.

16.134.2.3.2 `sbc_start_up_v2suc_t v2suc`

V2/VEXT start-up control.

Definition at line 1126 of file `sbc_uja116x_driver.h`.

16.134.2.4 `struct sbc_regulator_t`

Regulator control register structure. This structure set power distribution control, V2/VEXT configuration, set V1 reset threshold.

Implements : `sbc_regulator_t_Class`

Definition at line 1136 of file `sbc_uja116x_driver.h`.

## Data Fields

- [sbc\\_regulator\\_pdc\\_t pdc](#)
- [sbc\\_regulator\\_v2c\\_t v2c](#)
- [sbc\\_regulator\\_v1rtc\\_t v1rtc](#)

## Field Documentation

16.134.2.4.1 `sbc_regulator_pdc_t pdc`

Power distribution control.

Definition at line 1137 of file `sbc_uja116x_driver.h`.

16.134.2.4.2 `sbc_regulator_v1rtc_t v1rtc`

Set V1 reset threshold.

Definition at line 1139 of file `sbc_uja116x_driver.h`.

16.134.2.4.3 `sbc_regulator_v2c_t v2c`

V2/VEXT configuration.

Definition at line 1138 of file `sbc_uja116x_driver.h`.

16.134.2.5 `struct sbc_supply_evnt_t`

Supply event capture enable register structure. This structure enables or disables detection of V2/VEXT overvoltage, undervoltage and V1 undervoltage enable.

Implements : `sbc_supply_evnt_t_Class`

Definition at line 1149 of file `sbc_uja116x_driver.h`.

## Data Fields

- [sbc\\_supply\\_evnt\\_v2oe\\_t v2oe](#)
- [sbc\\_supply\\_evnt\\_v2ue\\_t v2ue](#)
- [sbc\\_supply\\_evnt\\_v1ue\\_t v1ue](#)

## Field Documentation

**16.134.2.5.1 sbc\_supply\_evnt\_v1ue\_t v1ue**

SV1 undervoltage enable.

Definition at line 1152 of file sbc\_uja116x\_driver.h.

**16.134.2.5.2 sbc\_supply\_evnt\_v2oe\_t v2oe**

V2/VEXT overvoltage enable.

Definition at line 1150 of file sbc\_uja116x\_driver.h.

**16.134.2.5.3 sbc\_supply\_evnt\_v2ue\_t v2ue**

V2/VEXT undervoltage enable.

Definition at line 1151 of file sbc\_uja116x\_driver.h.

**16.134.2.6 struct sbc\_sys\_evnt\_t**

System event capture enable register structure. This structure enables or disables overtemperature warning, SPI failure enable.

Implements : sbc\_sys\_evnt\_t\_Class

Definition at line 1162 of file sbc\_uja116x\_driver.h.

**Data Fields**

- [sbc\\_sys\\_evnt\\_otwe\\_t otwe](#)
- [sbc\\_sys\\_evnt\\_spife\\_t spife](#)

**Field Documentation****16.134.2.6.1 sbc\_sys\_evnt\_otwe\_t otwe**

Overtemperature warning enable.

Definition at line 1163 of file sbc\_uja116x\_driver.h.

**16.134.2.6.2 sbc\_sys\_evnt\_spife\_t spife**

SPI failure enable.

Definition at line 1164 of file sbc\_uja116x\_driver.h.

**16.134.2.7 struct sbc\_can\_ctr\_t**

CAN control register structure. This structure configure CAN peripheral behavior.

Implements : sbc\_can\_ctr\_t\_Class

Definition at line 1173 of file sbc\_uja116x\_driver.h.

**Data Fields**

- [sbc\\_can\\_cfdc\\_t cfcd](#)
- [sbc\\_can\\_pncok\\_t pncok](#)
- [sbc\\_can\\_cpnc\\_t cpnc](#)
- [sbc\\_can\\_cmc\\_t cmc](#)

**Field Documentation****16.134.2.7.1 sbc\_can\_cfdc\_t cfcd**

CAN FD control.

Definition at line 1174 of file sbc\_uja116x\_driver.h.

#### 16.134.2.7.2 `sbc_can_cmc_t cmc`

CAN mode control.

Definition at line 1179 of file sbc\_uja116x\_driver.h.

#### 16.134.2.7.3 `sbc_can_cpnc_t cpnc`

CAN partial. networking control.

Definition at line 1177 of file sbc\_uja116x\_driver.h.

#### 16.134.2.7.4 `sbc_can_pncok_t pncok`

CAN partial networking. configuration OK.

Definition at line 1175 of file sbc\_uja116x\_driver.h.

#### 16.134.2.8 `struct sbc_trans_evnt_t`

Transceiver event capture enable register structure. Can bus silence, Can failure and Can wake-up settings.

Implements : `sbc_trans_evnt_t_Class`

Definition at line 1188 of file sbc\_uja116x\_driver.h.

##### Data Fields

- [sbc\\_trans\\_evnt\\_cbse\\_t cbse](#)
- [sbc\\_trans\\_evnt\\_cfe\\_t cfe](#)
- [sbc\\_trans\\_evnt\\_cwe\\_t cwe](#)

##### Field Documentation

#### 16.134.2.8.1 `sbc_trans_evnt_cbse_t cbse`

CAN-bus silence enable.

Definition at line 1189 of file sbc\_uja116x\_driver.h.

#### 16.134.2.8.2 `sbc_trans_evnt_cfe_t cfe`

CAN failure enable.

Definition at line 1190 of file sbc\_uja116x\_driver.h.

#### 16.134.2.8.3 `sbc_trans_evnt_cwe_t cwe`

CAN wake-up enable.

Definition at line 1191 of file sbc\_uja116x\_driver.h.

#### 16.134.2.9 `struct sbc_frame_t`

Frame control register structure. The wake-up frame format, standard (11-bit) or extended (29-bit) identifier, is selected via bit IDE in the Frame control register.

Implements : `sbc_frame_t_Class`

Definition at line 1201 of file sbc\_uja116x\_driver.h.

##### Data Fields

- [sbc\\_frame\\_ctr\\_ide\\_t ide](#)

- [sbc\\_frame\\_ctr\\_pndm\\_t pndm](#)
- [sbc\\_frame\\_ctr\\_dlc\\_t dlc](#)

#### Field Documentation

##### 16.134.2.9.1 [sbc\\_frame\\_ctr\\_dlc\\_t dlc](#)

Number of data bytes expected.

Definition at line 1204 of file sbc\_uja116x\_driver.h.

##### 16.134.2.9.2 [sbc\\_frame\\_ctr\\_ide\\_t ide](#)

Identifier format.

Definition at line 1202 of file sbc\_uja116x\_driver.h.

##### 16.134.2.9.3 [sbc\\_frame\\_ctr\\_pndm\\_t pndm](#)

Partial networking data mask.

Definition at line 1203 of file sbc\_uja116x\_driver.h.

##### 16.134.2.10 [struct sbc\\_can\\_conf\\_t](#)

CAN configuration group structure. This structure configure CAN peripheral behavior.

Implements : [sbc\\_can\\_conf\\_t\\_Class](#)

Definition at line 1213 of file sbc\_uja116x\_driver.h.

#### Data Fields

- [sbc\\_can\\_ctr\\_t canConf](#)
- [sbc\\_trans\\_evnt\\_t canTransEvt](#)
- [sbc\\_dat\\_rate\\_t datRate](#)
- [sbc\\_identifier\\_t identif](#) [SBC\_UJA\_COUNT\_ID\_REG]
- [sbc\\_identif\\_mask\\_t mask](#) [SBC\_UJA\_COUNT\_MASK]
- [sbc\\_frame\\_t frame](#)
- [sbc\\_data\\_mask\\_t dataMask](#) [SBC\_UJA\_COUNT\_DMASK]

#### Field Documentation

##### 16.134.2.10.1 [sbc\\_can\\_ctr\\_t canConf](#)

CAN control register.

Definition at line 1214 of file sbc\_uja116x\_driver.h.

##### 16.134.2.10.2 [sbc\\_trans\\_evnt\\_t canTransEvt](#)

Transceiver event capture enable register.

Definition at line 1215 of file sbc\_uja116x\_driver.h.

##### 16.134.2.10.3 [sbc\\_data\\_mask\\_t dataMask](#)[SBC\_UJA\_COUNT\_DMASK]

Data mask 0 - 7 configuration.

Definition at line 1221 of file sbc\_uja116x\_driver.h.

##### 16.134.2.10.4 [sbc\\_dat\\_rate\\_t datRate](#)

CAN data rate selection.

Definition at line 1217 of file sbc\_uja116x\_driver.h.



**16.134.2.10.5 sbc\_frame\_t frame**

Frame control register.

Definition at line 1220 of file sbc\_uja116x\_driver.h.

**16.134.2.10.6 sbc\_identifier\_t identif[SBC\_UJA\_COUNT\_ID\_REG]**

ID registers.

Definition at line 1218 of file sbc\_uja116x\_driver.h.

**16.134.2.10.7 sbc\_identif\_mask\_t mask[SBC\_UJA\_COUNT\_MASK]**

ID mask registers.

Definition at line 1219 of file sbc\_uja116x\_driver.h.

**16.134.2.11 struct sbc\_wake\_t**

WAKE pin event capture enable register structure. Local wake-up is enabled via bits WPRE and WPFE in the WAKE pin event capture enable register. A wake-up event is triggered by a LOW-to-HIGH (if WPRE = 1) and/or a HIGH-to-LOW (if WPFE = 1) transition on the WAKE pin. This arrangement allows for maximum flexibility when designing a local wake-up circuit. In applications that do not use the local wake-up facility, local wake-up should be disabled and the WAKE pin connected to GND.

Implements : sbc\_wake\_t\_Class

Definition at line 1236 of file sbc\_uja116x\_driver.h.

**Data Fields**

- [sbc\\_wake\\_en\\_wpre\\_t wpre](#)
- [sbc\\_wake\\_en\\_wpfe\\_t wpfe](#)

**Field Documentation****16.134.2.11.1 sbc\_wake\_en\_wpfe\_t wpfe**

WAKE pin falling-edge enable.

Definition at line 1238 of file sbc\_uja116x\_driver.h.

**16.134.2.11.2 sbc\_wake\_en\_wpre\_t wpre**

WAKE pin rising-edge enable.

Definition at line 1237 of file sbc\_uja116x\_driver.h.

**16.134.2.12 struct sbc\_regulator\_ctr\_t**

Regulator control register group. This structure is group of regulator settings.

Implements : sbc\_regulator\_ctr\_t\_Class

Definition at line 1247 of file sbc\_uja116x\_driver.h.

**Data Fields**

- [sbc\\_regulator\\_t regulator](#)
- [sbc\\_supply\\_evnt\\_t supplyEvnt](#)

**Field Documentation**

**16.134.2.12.1 sbc\_regulator\_t regulator**

Regulator control register.

Definition at line 1248 of file sbc\_uja116x\_driver.h.

**16.134.2.12.2 sbc\_supply\_evnt\_t supplyEvt**

Supply event capture enable register.

Definition at line 1249 of file sbc\_uja116x\_driver.h.

**16.134.2.13 struct sbc\_int\_config\_t**

Init configuration structure. This structure is used for initialization of sbc.

Implements : sbc\_int\_config\_t\_Class

Definition at line 1259 of file sbc\_uja116x\_driver.h.

**Data Fields**

- [sbc\\_regulator\\_ctr\\_t regulatorCtr](#)
- [sbc\\_wtdog\\_ctr\\_t watchdog](#)
- [sbc\\_mode\\_mc\\_t mode](#)
- [sbc\\_fail\\_safe\\_lhc\\_t lhc](#)
- [sbc\\_sys\\_evnt\\_t sysEvt](#)
- [sbc\\_lock\\_t lockMask](#)
- [sbc\\_can\\_conf\\_t can](#)
- [sbc\\_wake\\_t wakePin](#)

**Field Documentation****16.134.2.13.1 sbc\_can\_conf\_t can**

CAN configuration group.

Definition at line 1267 of file sbc\_uja116x\_driver.h.

**16.134.2.13.2 sbc\_fail\_safe\_lhc\_t lhc**

LIMP home control.

Definition at line 1263 of file sbc\_uja116x\_driver.h.

**16.134.2.13.3 sbc\_lock\_t lockMask**

Lock control register.

Definition at line 1266 of file sbc\_uja116x\_driver.h.

**16.134.2.13.4 sbc\_mode\_mc\_t mode**

Mode control register.

Definition at line 1262 of file sbc\_uja116x\_driver.h.

**16.134.2.13.5 sbc\_regulator\_ctr\_t regulatorCtr**

Regulator control register group.

Definition at line 1260 of file sbc\_uja116x\_driver.h.

**16.134.2.13.6 sbc\_sys\_evnt\_t sysEvt**

System event capture enable registers.

Definition at line 1264 of file sbc\_uja116x\_driver.h.

#### 16.134.2.13.7 `sbc_wake_t` wakePin

WAKE pin event capture enable register.

Definition at line 1268 of file sbc\_uja116x\_driver.h.

#### 16.134.2.13.8 `sbc_wtdog_ctr_t` watchdog

Watchdog control register.

Definition at line 1261 of file sbc\_uja116x\_driver.h.

#### 16.134.2.14 `struct sbc_factories_conf_t`

Factory configuration structure. It contains Start-up control register and SBC configuration control register. This is non-volatile memory with limited write access. The MTPNV cells can be reprogrammed a maximum of 200 times (Ncy(W)MTP; Bit NVMPs in the MTPNV status register indicates whether the non-volatile cells can be reprogrammed. This register also contains a write counter, WRCNTS, that is incremented each time the MTPNV cells are reprogrammed (up to a maximum value of 111111; there is no overflow; performing a factory reset also increments the counter). This counter is provided for information purposes only; reprogramming will not be rejected when it reaches its maximum value. Factory preset values are restored if the following conditions apply continuously for at least td(MTPNV) during battery power-up: pin RSTN is held LOW, CANH is pulled up to VBAT, CANL is pulled down to GND. After the factory preset values have been restored, the SBC performs a system reset and enters Forced normal Mode. Since the CAN-bus is clamped dominant, pin RXDC is forced LOW. Pin RXD is forced HIGH during the factory preset restore process (td(MTPNV)). A falling edge on RXD caused by bit PO being set after power-on indicates that the factory preset process has been completed. Note that the write counter, WRCNTS, in the MTPNV status register is incremented every time the factory presets are restored.

Implements : `sbc_factories_conf_t` Class

Definition at line 1298 of file sbc\_uja116x\_driver.h.

#### Data Fields

- [sbc\\_start\\_up\\_t](#) startUp
- [sbc\\_sbc\\_t](#) control

#### Field Documentation

##### 16.134.2.14.1 `sbc_sbc_t` control

SBC configuration control register. Note that this register is located in the non-volatile memory area.

Definition at line 1300 of file sbc\_uja116x\_driver.h.

##### 16.134.2.14.2 `sbc_start_up_t` startUp

Start-up control register.

Definition at line 1299 of file sbc\_uja116x\_driver.h.

#### 16.134.2.15 `struct sbc_main_status_t`

Main status register structure. The Main status register can be accessed to monitor the status of the overtemperature warning flag and to determine whether the UJA116xA has entered Normal mode after initial power-up. It also indicates the source of the most recent reset event.

Implements : `sbc_main_status_t` Class

Definition at line 1314 of file sbc\_uja116x\_driver.h.

**Data Fields**

- [sbc\\_main\\_otws\\_t otws](#)
- [sbc\\_main\\_nms\\_t nms](#)
- [sbc\\_main\\_rss\\_t rss](#)

**Field Documentation****16.134.2.15.1 sbc\_main\_nms\_t nms**

Normal mode status.

Definition at line 1316 of file sbc\_uja116x\_driver.h.

**16.134.2.15.2 sbc\_main\_otws\_t otws**

Overtemperature warning status.

Definition at line 1315 of file sbc\_uja116x\_driver.h.

**16.134.2.15.3 sbc\_main\_rss\_t rss**

Reset source status.

Definition at line 1317 of file sbc\_uja116x\_driver.h.

**16.134.2.16 struct sbc\_wtdog\_status\_t**

Watchdog status register structure. Information on the status of the watchdog is available from the Watchdog status register. This register also indicates whether Forced Normal and Software Development modes are active.

Implements : [sbc\\_wtdog\\_status\\_t\\_Class](#)

Definition at line 1328 of file sbc\_uja116x\_driver.h.

**Data Fields**

- [sbc\\_wtdog\\_stat\\_fnms\\_t fnms](#)
- [sbc\\_wtdog\\_stat\\_sdms\\_t sdms](#)
- [sbc\\_wtdog\\_stat\\_wds\\_t wds](#)

**Field Documentation****16.134.2.16.1 sbc\_wtdog\_stat\_fnms\_t fnms**

Forced Normal mode status.

Definition at line 1329 of file sbc\_uja116x\_driver.h.

**16.134.2.16.2 sbc\_wtdog\_stat\_sdms\_t sdms**

Software Development mode status.

Definition at line 1330 of file sbc\_uja116x\_driver.h.

**16.134.2.16.3 sbc\_wtdog\_stat\_wds\_t wds**

Watchdog status.

Definition at line 1331 of file sbc\_uja116x\_driver.h.

**16.134.2.17 struct sbc\_supply\_status\_t**

Supply voltage status register structure. V2/VEXT and V1 undervoltage and overvoltage status.

Implements : [sbc\\_supply\\_status\\_t\\_Class](#)

Definition at line 1340 of file sbc\_uja116x\_driver.h.

#### Data Fields

- [sbc\\_supply\\_stat\\_v2s\\_t v2s](#)
- [sbc\\_supply\\_stat\\_v1s\\_t v1s](#)

#### Field Documentation

##### 16.134.2.17.1 sbc\_supply\_stat\_v1s\_t v1s

V1 status.

Definition at line 1342 of file sbc\_uja116x\_driver.h.

##### 16.134.2.17.2 sbc\_supply\_stat\_v2s\_t v2s

V2/VEXT status.

Definition at line 1341 of file sbc\_uja116x\_driver.h.

##### 16.134.2.18 struct sbc\_trans\_stat\_t

Transceiver status register structure. There are stored CAN transceiver statuses.

Implements : sbc\_trans\_stat\_t\_Class

Definition at line 1351 of file sbc\_uja116x\_driver.h.

#### Data Fields

- [sbc\\_trans\\_stat\\_cts\\_t cts](#)
- [sbc\\_trans\\_stat\\_cpnerr\\_t cpnerr](#)
- [sbc\\_trans\\_stat\\_cpns\\_t cpns](#)
- [sbc\\_trans\\_stat\\_coscs\\_t coscs](#)
- [sbc\\_trans\\_stat\\_cbss\\_t cbss](#)
- [sbc\\_trans\\_stat\\_vcs\\_t vcs](#)
- [sbc\\_trans\\_stat\\_cfs\\_t cfs](#)

#### Field Documentation

##### 16.134.2.18.1 sbc\_trans\_stat\_cbss\_t cbss

CAN-bus silence status.

Definition at line 1356 of file sbc\_uja116x\_driver.h.

##### 16.134.2.18.2 sbc\_trans\_stat\_cfs\_t cfs

CAN failure status.

Definition at line 1358 of file sbc\_uja116x\_driver.h.

##### 16.134.2.18.3 sbc\_trans\_stat\_coscs\_t coscs

CAN oscillator status.

Definition at line 1355 of file sbc\_uja116x\_driver.h.

##### 16.134.2.18.4 sbc\_trans\_stat\_cpnerr\_t cpnerr

CAN partial networking error.

Definition at line 1353 of file sbc\_uja116x\_driver.h.

**16.134.2.18.5 sbc\_trans\_stat\_cpns\_t cpns**

CAN partial networking status.

Definition at line 1354 of file sbc\_uja116x\_driver.h.

**16.134.2.18.6 sbc\_trans\_stat\_cts\_t cts**

CAN transceiver status.

Definition at line 1352 of file sbc\_uja116x\_driver.h.

**16.134.2.18.7 sbc\_trans\_stat\_vcs\_t vcs**

VCAN status.

Definition at line 1357 of file sbc\_uja116x\_driver.h.

**16.134.2.19 struct sbc\_gl\_evnt\_stat\_t**

Global event status register. The microcontroller can monitor events via the event status registers. An extra status register, the Global event status register, is provided to help speed up software polling routines. By polling the Global event status register, the microcontroller can quickly determine the type of event captured (system, supply, transceiver or WAKE pin) and then query the relevant event status register.

Implements : sbc\_gl\_evnt\_stat\_t\_Class

Definition at line 1372 of file sbc\_uja116x\_driver.h.

**Data Fields**

- [sbc\\_gl\\_evnt\\_stat\\_wpe\\_t wpe](#)
- [sbc\\_gl\\_evnt\\_stat\\_trxe\\_t trxe](#)
- [sbc\\_gl\\_evnt\\_stat\\_supe\\_t supe](#)
- [sbc\\_gl\\_evnt\\_stat\\_syse\\_t syse](#)

**Field Documentation****16.134.2.19.1 sbc\_gl\_evnt\_stat\_supe\_t supe**

Supply event.

Definition at line 1375 of file sbc\_uja116x\_driver.h.

**16.134.2.19.2 sbc\_gl\_evnt\_stat\_syse\_t syse**

System event.

Definition at line 1376 of file sbc\_uja116x\_driver.h.

**16.134.2.19.3 sbc\_gl\_evnt\_stat\_trxe\_t trxe**

Transceiver event.

Definition at line 1374 of file sbc\_uja116x\_driver.h.

**16.134.2.19.4 sbc\_gl\_evnt\_stat\_wpe\_t wpe**

WAKE pin event.

Definition at line 1373 of file sbc\_uja116x\_driver.h.

**16.134.2.20 struct sbc\_sys\_evnt\_stat\_t**

System event status register. Wake-up and interrupt event diagnosis in the UJA116xA is intended to provide the microcontroller with information on the status of a range of features and functions. This information is stored in the

event status registers and is signaled on pin RXD, if enabled.

Implements : `sbc_sys_evnt_stat_t_Class`

Definition at line 1388 of file `sbc_uja116x_driver.h`.

#### Data Fields

- [sbc\\_sys\\_evnt\\_stat\\_po\\_t po](#)
- [sbc\\_sys\\_evnt\\_stat\\_otw\\_t otw](#)
- [sbc\\_sys\\_evnt\\_stat\\_spif\\_t spif](#)
- [sbc\\_sys\\_evnt\\_stat\\_wdf\\_t wdf](#)

#### Field Documentation

##### 16.134.2.20.1 `sbc_sys_evnt_stat_otw_t otw`

Transceiver event, overtemperature warning

Definition at line 1390 of file `sbc_uja116x_driver.h`.

##### 16.134.2.20.2 `sbc_sys_evnt_stat_po_t po`

Power-on.

Definition at line 1389 of file `sbc_uja116x_driver.h`.

##### 16.134.2.20.3 `sbc_sys_evnt_stat_spif_t spif`

SPI failure.

Definition at line 1392 of file `sbc_uja116x_driver.h`.

##### 16.134.2.20.4 `sbc_sys_evnt_stat_wdf_t wdf`

Watchdog failure.

Definition at line 1393 of file `sbc_uja116x_driver.h`.

##### 16.134.2.21 `struct sbc_sup_evnt_stat_t`

Supply event status register.

Implements : `sbc_sup_evnt_stat_t_Class`

Definition at line 1401 of file `sbc_uja116x_driver.h`.

#### Data Fields

- [sbc\\_sup\\_evnt\\_stat\\_v2o\\_t v2o](#)
- [sbc\\_sup\\_evnt\\_stat\\_v2u\\_t v2u](#)
- [sbc\\_sup\\_evnt\\_stat\\_v1u\\_t v1u](#)

#### Field Documentation

##### 16.134.2.21.1 `sbc_sup_evnt_stat_v1u_t v1u`

V1 undervoltage.

Definition at line 1404 of file `sbc_uja116x_driver.h`.

##### 16.134.2.21.2 `sbc_sup_evnt_stat_v2o_t v2o`

V2/VEXT overvoltage.

Definition at line 1402 of file `sbc_uja116x_driver.h`.

**16.134.2.21.3 sbc\_sup\_evnt\_stat\_v2u\_t v2u**

V2/VEXT undervoltage.

Definition at line 1403 of file sbc\_uja116x\_driver.h.

**16.134.2.22 struct sbc\_trans\_evnt\_stat\_t**

Transceiver event status register.

Implements : sbc\_trans\_evnt\_stat\_t\_Class

Definition at line 1412 of file sbc\_uja116x\_driver.h.

**Data Fields**

- [sbc\\_trans\\_evnt\\_stat\\_pnfde\\_t pnfde](#)
- [sbc\\_trans\\_evnt\\_stat\\_cbs\\_t cbs](#)
- [sbc\\_trans\\_evnt\\_stat\\_cf\\_t cf](#)
- [sbc\\_trans\\_evnt\\_stat\\_cw\\_t cw](#)

**Field Documentation****16.134.2.22.1 sbc\_trans\_evnt\_stat\_cbs\_t cbs**

CAN-bus status.

Definition at line 1415 of file sbc\_uja116x\_driver.h.

**16.134.2.22.2 sbc\_trans\_evnt\_stat\_cf\_t cf**

CAN failure.

Definition at line 1416 of file sbc\_uja116x\_driver.h.

**16.134.2.22.3 sbc\_trans\_evnt\_stat\_cw\_t cw**

CAN wake-up.

Definition at line 1417 of file sbc\_uja116x\_driver.h.

**16.134.2.22.4 sbc\_trans\_evnt\_stat\_pnfde\_t pnfde**

Partial networking frame detection error.

Definition at line 1413 of file sbc\_uja116x\_driver.h.

**16.134.2.23 struct sbc\_wake\_evnt\_stat\_t**

WAKE pin event status register.

Implements : sbc\_wake\_evnt\_stat\_t\_Class

Definition at line 1425 of file sbc\_uja116x\_driver.h.

**Data Fields**

- [sbc\\_wake\\_evnt\\_stat\\_wpr\\_t wpr](#)
- [sbc\\_wake\\_evnt\\_stat\\_wpf\\_t wpf](#)

**Field Documentation****16.134.2.23.1 sbc\_wake\_evnt\_stat\_wpf\_t wpf**

WAKE pin falling edge.

Definition at line 1427 of file sbc\_uja116x\_driver.h.



**16.134.2.23.2 sbc\_wake\_evnt\_stat\_wpr\_t wpr**

WAKE pin rising edge.

Definition at line 1426 of file sbc\_uja116x\_driver.h.

**16.134.2.24 struct sbc\_evn\_capt\_t**

Event capture registers structure. This structure contains Global event status, System event status, Supply event status, Transceiver event status, WAKE pin event status.

Implements : sbc\_evn\_capt\_t\_Class

Definition at line 1437 of file sbc\_uja116x\_driver.h.

**Data Fields**

- [sbc\\_gl\\_evnt\\_stat\\_t glEvt](#)
- [sbc\\_sys\\_evnt\\_stat\\_t sysEvt](#)
- [sbc\\_sup\\_evnt\\_stat\\_t supEvt](#)
- [sbc\\_trans\\_evnt\\_stat\\_t transEvt](#)
- [sbc\\_wake\\_evnt\\_stat\\_t wakePinEvt](#)

**Field Documentation****16.134.2.24.1 sbc\_gl\_evnt\_stat\_t glEvt**

Global event status.

Definition at line 1438 of file sbc\_uja116x\_driver.h.

**16.134.2.24.2 sbc\_sup\_evnt\_stat\_t supEvt**

Supply event status.

Definition at line 1440 of file sbc\_uja116x\_driver.h.

**16.134.2.24.3 sbc\_sys\_evnt\_stat\_t sysEvt**

System event status.

Definition at line 1439 of file sbc\_uja116x\_driver.h.

**16.134.2.24.4 sbc\_trans\_evnt\_stat\_t transEvt**

Transceiver event status.

Definition at line 1441 of file sbc\_uja116x\_driver.h.

**16.134.2.24.5 sbc\_wake\_evnt\_stat\_t wakePinEvt**

WAKE pin event status.

Definition at line 1442 of file sbc\_uja116x\_driver.h.

**16.134.2.25 struct sbc\_mtpnv\_stat\_t**

MTPNV status register. The MTPNV cells can be reprogrammed a maximum of 200 times (Ncy(W)MTP). Bit N↔ VMPS in the MTPNV status register indicates whether the non-volatile cells can be reprogrammed. This register also contains a write counter, WRCNTS, that is incremented each time the MTPNV cells are reprogrammed (up to a maximum value of 111111; there is no overflow; performing a factory reset also increments the counter). This counter is provided for information purposes only; reprogramming will not be rejected when it reaches its maximum value.

Implements : sbc\_mtpnv\_stat\_t\_Class

Definition at line 1458 of file sbc\_uja116x\_driver.h.

#### Data Fields

- [sbc\\_mtpnv\\_stat\\_wrcnts\\_t wrcnts](#)
- [sbc\\_mtpnv\\_stat\\_eccs\\_t eccs](#)
- [sbc\\_mtpnv\\_stat\\_nvmps\\_t nvmps](#)

#### Field Documentation

##### 16.134.2.25.1 **sbc\_mtpnv\_stat\_eccs\_t eccs**

Error correction code status.

Definition at line 1460 of file sbc\_uja116x\_driver.h.

##### 16.134.2.25.2 **sbc\_mtpnv\_stat\_nvmps\_t nvmps**

Non-volatile memory programming status.

Definition at line 1461 of file sbc\_uja116x\_driver.h.

##### 16.134.2.25.3 **sbc\_mtpnv\_stat\_wrcnts\_t wrcnts**

Write counter status.

Definition at line 1459 of file sbc\_uja116x\_driver.h.

##### 16.134.2.26 **struct sbc\_status\_group\_t**

Status group structure. All statuses of SBC are stored in this structure.

Implements : [sbc\\_status\\_group\\_t\\_Class](#)

Definition at line 1472 of file sbc\_uja116x\_driver.h.

#### Data Fields

- [sbc\\_main\\_status\\_t mainS](#)
- [sbc\\_wtdog\\_status\\_t wtdog](#)
- [sbc\\_supply\\_status\\_t supply](#)
- [sbc\\_trans\\_stat\\_t trans](#)
- [sbc\\_wake\\_stat\\_wpvs\\_t wakePin](#)
- [sbc\\_evn\\_capt\\_t events](#)

#### Field Documentation

##### 16.134.2.26.1 **sbc\_evn\_capt\_t events**

Event capture registers.

Definition at line 1478 of file sbc\_uja116x\_driver.h.

##### 16.134.2.26.2 **sbc\_main\_status\_t mainS**

Main status.

Definition at line 1473 of file sbc\_uja116x\_driver.h.

##### 16.134.2.26.3 **sbc\_supply\_status\_t supply**

Supply voltage status.

Definition at line 1475 of file sbc\_uja116x\_driver.h.

**16.134.2.26.4 sbc\_trans\_stat\_t trans**

Transceiver status.

Definition at line 1476 of file sbc\_uja116x\_driver.h.

**16.134.2.26.5 sbc\_wake\_stat\_wpvs\_t wakePin**

WAKE pin status.

Definition at line 1477 of file sbc\_uja116x\_driver.h.

**16.134.2.26.6 sbc\_wtdog\_status\_t wtdog**

Watchdog status.

Definition at line 1474 of file sbc\_uja116x\_driver.h.

**16.134.3 Macro Definition Documentation****16.134.3.1 #define SBC\_UJA\_COUNT\_DMASK 8U**

Definition at line 41 of file sbc\_uja116x\_driver.h.

**16.134.3.2 #define SBC\_UJA\_COUNT\_ID\_REG 4U**

Definition at line 39 of file sbc\_uja116x\_driver.h.

**16.134.3.3 #define SBC\_UJA\_COUNT\_MASK 4U**

Definition at line 40 of file sbc\_uja116x\_driver.h.

**16.134.3.4 #define SBC\_UJA\_TIMEOUT 1000U**

Timeout for the transfer in milliseconds. If the transfer takes longer than this time, the transfer is aborted and LPSPi\_STATUS\_SBC\_UJA\_TIMEOUT error is reported.

Definition at line 33 of file sbc\_uja116x\_driver.h.

**16.134.4 Typedef Documentation****16.134.4.1 typedef uint8\_t sbc\_data\_mask\_t**

Data mask registers. The data field indicates the nodes to be woken up. Within the data field, groups of nodes can be predefined and associated with bits in a data mask. By comparing the incoming data field with the data mask, multiple groups of nodes can be woken up simultaneously with a single wake-up message.

Implements : sbc\_data\_mask\_t\_Class

Definition at line 706 of file sbc\_uja116x\_driver.h.

**16.134.4.2 typedef uint8\_t sbc\_fail\_safe\_rcc\_t**

Fail-safe control register, reset counter control (0x02). incremented every time the SBC enters Reset mode while FNMC = 0; RCC overflows from 11 to 00; default at power-on is 00.

Implements : sbc\_fail\_safe\_rcc\_t\_Class

Definition at line 195 of file sbc\_uja116x\_driver.h.

**16.134.4.3 typedef uint8\_t sbc\_frame\_ctr\_dlc\_t**

Frame control register, number of data bytes expected in a CAN frame (0x2F).

Implements : `sbc_frame_ctr_dlc_t_Class`

Definition at line 695 of file `sbc_uja116x_driver.h`.

#### 16.134.4.4 `typedef uint8_t sbc_identif_mask_t`

ID mask registers (0x2B to 0x2E). The identifier mask is defined in the ID mask registers, where a 1 means dont care.

Implements : `sbc_identif_mask_t_Class`

Definition at line 661 of file `sbc_uja116x_driver.h`.

#### 16.134.4.5 `typedef uint8_t sbc_identifier_t`

ID registers, identifier format (0x27 to 0x2A). A valid WUF identifier is defined and stored in the ID registers. An ID mask can be defined to allow a group of identifiers to be recognized as valid by an individual node.

Implements : `sbc_identifier_t_Class`

Definition at line 652 of file `sbc_uja116x_driver.h`.

#### 16.134.4.6 `typedef uint8_t sbc_mtpnv_stat_wrcnts_t`

MTPNV status register, write counter status (0x70). 6-bits - contains the number of times the MTPNV cells were reprogrammed.

Implements : `sbc_mtpnv_stat_wrcnts_t_Class`

Definition at line 967 of file `sbc_uja116x_driver.h`.

### 16.134.5 Enumeration Type Documentation

#### 16.134.5.1 `enum sbc_can_cfdc_t`

CAN control register, CAN FD control (0x20).

Implements : `sbc_can_cfdc_t_Class`

#### Enumerator

**`SBC_UJA_CAN_CFDC_DIS`** CAN FD tolerance disabled.

**`SBC_UJA_CAN_CFDC_EN`** CAN FD tolerance enabled.

Definition at line 460 of file `sbc_uja116x_driver.h`.

#### 16.134.5.2 `enum sbc_can_cmc_t`

CAN control register, CAN mode control (0x20).

Implements : `sbc_can_cmc_t_Class`

#### Enumerator

**`SBC_UJA_CAN_CMC_OFMODE`** Offline mode.

**`SBC_UJA_CAN_CMC_ACMODE_DA`** Active mode (when the SBC is in Normal mode); CAN supply under-voltage detection active.

**`SBC_UJA_CAN_CMC_ACMODE_DD`** Active mode (when the SBC is in Normal mode); CAN supply under-voltage detection disabled.

**`SBC_UJA_CAN_CMC_LISTEN`** Listen-only mode.

Definition at line 496 of file `sbc_uja116x_driver.h`.

16.134.5.3 enum **sbc\_can\_cpnc\_t**

CAN control register, CAN partial networking control (0x20).

Implements : `sbc_can_cpnc_t_Class`

## Enumerator

**SBC\_UJA\_CAN\_CPNC\_DIS** Disable CAN selective wake-up.

**SBC\_UJA\_CAN\_CPNC\_EN** Enable CAN selective wake-up.

Definition at line 484 of file `sbc_uja116x_driver.h`.

16.134.5.4 enum **sbc\_can\_pncok\_t**

CAN control register, CAN partial networking configuration OK (0x20).

Implements : `sbc_can_pncok_t_Class`

## Enumerator

**SBC\_UJA\_CAN\_PNCOK\_DIS** Partial networking register configuration invalid (wake-up via standard wake-up pattern only).

**SBC\_UJA\_CAN\_PNCOK\_EN** Partial networking registers configured successfully.

Definition at line 472 of file `sbc_uja116x_driver.h`.

16.134.5.5 enum **sbc\_dat\_rate\_t**

Data rate register, CAN data rate selection (0x26). CAN partial networking configuration registers. Dedicated registers are provided for configuring CAN partial networking.

Implements : `sbc_dat_rate_t_Class`

## Enumerator

**SBC\_UJA\_DAT\_RATE\_CDR\_50KB** 50 kbit/s.

**SBC\_UJA\_DAT\_RATE\_CDR\_100KB** 100 kbit/s.

**SBC\_UJA\_DAT\_RATE\_CDR\_125KB** 125 kbit/s.

**SBC\_UJA\_DAT\_RATE\_CDR\_250KB** 250 kbit/s.

**SBC\_UJA\_DAT\_RATE\_CDR\_500KB** 500 kbit/s.

**SBC\_UJA\_DAT\_RATE\_CDR\_1000KB** 1000 kbit/s.

Definition at line 635 of file `sbc_uja116x_driver.h`.

16.134.5.6 enum **sbc\_fail\_safe\_lhc\_t**

Fail-safe control register, LIMP home control (0x02). The dedicated LIMP pin can be used to enable so called limp home hardware in the event of a serious ECU failure. Detectable failure conditions include SBC overtemperature events, loss of watchdog service, short-circuits on pins RSTN or V1 and user-initiated or external reset events. The LIMP pin is a battery-robust, active-LOW, open-drain output. The LIMP pin can also be forced LOW by setting bit LHC in the Fail-safe control register.

Implements : `sbc_fail_safe_lhc_t_Class`

## Enumerator

**SBC\_UJA\_FAIL\_SAFE\_LHC\_FLOAT** LIMP pin is floating.

**SBC\_UJA\_FAIL\_SAFE\_LHC\_LOW** LIMP pin is driven LOW.

Definition at line 183 of file `sbc_uja116x_driver.h`.

#### 16.134.5.7 enum sbc\_frame\_ctr\_ide\_t

Frame control register, identifier format (0x2F). The wake-up frame format, standard (11-bit) or extended (29-bit) identifier, is selected via bit IDE in the Frame control register.

Implements : sbc\_frame\_ctr\_ide\_t\_Class

##### Enumerator

**SBC\_UJA\_FRAME\_CTR\_IDE\_11B** Standard frame format (11-bit).

**SBC\_UJA\_FRAME\_CTR\_IDE\_29B** Extended frame format (29-bit).

Definition at line 670 of file sbc\_uja116x\_driver.h.

#### 16.134.5.8 enum sbc\_frame\_ctr\_pndm\_t

Frame control register, partial networking data mask (0x2F).

Implements : sbc\_frame\_ctr\_pndm\_t\_Class

##### Enumerator

**SBC\_UJA\_FRAME\_CTR\_PNDM\_DCARE** Data length code and data field are do not care for wake-up.

**SBC\_UJA\_FRAME\_CTR\_PNDM\_EVAL** Data length code and data field are evaluated at wake-up.

Definition at line 682 of file sbc\_uja116x\_driver.h.

#### 16.134.5.9 enum sbc\_gl\_evnt\_stat\_supe\_t

Global event status register, supply event (0x60).

Implements : sbc\_gl\_evnt\_stat\_supe\_t\_Class

##### Enumerator

**SBC\_UJA\_GL\_EVNT\_STAT\_SUPE\_NO** No pending supply event.

**SBC\_UJA\_GL\_EVNT\_STAT\_SUPE** Supply event pending at address 0x62 .

Definition at line 773 of file sbc\_uja116x\_driver.h.

#### 16.134.5.10 enum sbc\_gl\_evnt\_stat\_syse\_t

Global event status register, system event (0x60).

Implements : sbc\_gl\_evnt\_stat\_syse\_t\_Class

##### Enumerator

**SBC\_UJA\_GL\_EVNT\_STAT\_SYSE\_NO** No pending system event.

**SBC\_UJA\_GL\_EVNT\_STAT\_SYSE** System event pending at address 0x61.

Definition at line 785 of file sbc\_uja116x\_driver.h.

#### 16.134.5.11 enum sbc\_gl\_evnt\_stat\_trxe\_t

Global event status register, transceiver event (0x60).

Implements : sbc\_gl\_evnt\_stat\_trxe\_t\_Class

##### Enumerator

**SBC\_UJA\_GL\_EVNT\_STAT\_TRXE\_NO** No pending transceiver event.

**SBC\_UJA\_GL\_EVNT\_STAT\_TRXE** Transceiver event pending at address 0x63.

Definition at line 761 of file sbc\_uja116x\_driver.h.

16.134.5.12 enum `sbc_gl_evnt_stat_wpe_t`

Global event status register, WAKE pin event (0x60).

Implements : `sbc_gl_evnt_stat_wpe_t_Class`

## Enumerator

***SBC\_UJA\_GL\_EVNT\_STAT\_WPE\_NO*** No pending WAKE pin event.

***SBC\_UJA\_GL\_EVNT\_STAT\_WPE*** WAKE pin event pending at address 0x64.

Definition at line 749 of file `sbc_uja116x_driver.h`.

16.134.5.13 enum `sbc_lock_t`

Lock control(0x0A). Sections of the register address area can be write-protected to protect against unintended modifications. This facility only protects locked bits from being modified via the SPI and will not prevent the UJA116xA updating status registers etc.

Implements : `sbc_lock_t_Class`

## Enumerator

***LK0C*** Lock control 0: address area 0x06 to 0x09 - general-purpose memory macros. Lock control 1: address area 0x10 to 0x1F - regulator control macros.

***LK1C*** Lock control 2: address area 0x20 to 0x2F - transceiver control macros.

***LK2C*** Lock control 3: address area 0x30 to 0x3F - unused register range macros.

***LK3C*** Lock control 4: address area 0x40 to 0x4F - WAKE pin control macros.

***LK4C*** Lock control 5: address area 0x50 to 0x5F.

***LK5C*** Lock control 6: address area 0x68 to 0x6F macros.

***LK6C*** Lock control All: address area 0x10 to 0x6F macros.

***LKAC***

Definition at line 317 of file `sbc_uja116x_driver.h`.

16.134.5.14 enum `sbc_main_nms_t`

Main status register, normal mode status (0x03).

Implements : `sbc_main_nms_t_Class`

## Enumerator

***SBC\_UJA\_MAIN\_NMS\_NORMAL*** UJA116xA has entered Normal mode (after power-up)

***SBC\_UJA\_MAIN\_NMS\_PWR\_UP*** UJA116xA has powered up but has not yet switched to Normal mode.

Definition at line 214 of file `sbc_uja116x_driver.h`.

16.134.5.15 enum `sbc_main_otws_t`

Main status register, Overtemperature warning status (0x03).

Implements : `sbc_main_otws_t_Class`

## Enumerator

***SBC\_UJA\_MAIN\_OTWS\_BELOW*** IC temperature below overtemperature warning threshold.

***SBC\_UJA\_MAIN\_OTWS\_ABOVE*** IC temperature above overtemperature warning threshold.

Definition at line 202 of file `sbc_uja116x_driver.h`.

## 16.134.5.16 enum sbc\_main\_rss\_t

Main status register, Reset source status (0x03).

Implements : sbc\_main\_rss\_t\_Class

## Enumerator

- SBC\_UJA\_MAIN\_RSS\_OFF\_MODE** Left Off mode (power-on).
- SBC\_UJA\_MAIN\_RSS\_CAN\_WAKEUP** CAN wake-up in Sleep mode.
- SBC\_UJA\_MAIN\_RSS\_SLP\_WAKEUP** Wake-up via WAKE pin in Sleep mode.
- SBC\_UJA\_MAIN\_RSS\_OVF\_SLP** Watchdog overflow in Sleep mode (Timeout mode).
- SBC\_UJA\_MAIN\_RSS\_DIAG\_WAKEUP** Diagnostic wake-up in Sleep mode
- SBC\_UJA\_MAIN\_RSS\_WATCH\_TRIG** Watchdog triggered too early (Window mode).
- SBC\_UJA\_MAIN\_RSS\_WATCH\_OVF** Watchdog overflow (Window mode or Timeout mode with WDF = 1)
- SBC\_UJA\_MAIN\_RSS\_ILLEG\_WATCH** Illegal watchdog mode control access.
- SBC\_UJA\_MAIN\_RSS\_RSTN\_PULDW** RSTN pulled down externally.
- SBC\_UJA\_MAIN\_RSS\_LFT\_OVERTM** Left Overtemp mode.
- SBC\_UJA\_MAIN\_RSS\_V1\_UNDERV** V1 undervoltage.
- SBC\_UJA\_MAIN\_RSS\_ILLEG\_SLP** Illegal Sleep mode command received.
- SBC\_UJA\_MAIN\_RSS\_WAKE\_SLP** Wake-up from Sleep mode due to a frame detect error

Definition at line 226 of file sbc\_uja116x\_driver.h.

## 16.134.5.17 enum sbc\_mode\_mc\_t

Mode control register, mode control (0x01)

Implements : sbc\_mode\_mc\_t\_Class

## Enumerator

- SBC\_UJA\_MODE\_MC\_SLEEP** Sleep mode.
- SBC\_UJA\_MODE\_MC\_STANDBY** Standby mode.
- SBC\_UJA\_MODE\_MC\_NORMAL** Normal mode.

Definition at line 165 of file sbc\_uja116x\_driver.h.

## 16.134.5.18 enum sbc\_mtpnv\_stat\_eccs\_t

MTPNV status register, error correction code status (0x70).

Implements : sbc\_mtpnv\_stat\_eccs\_t\_Class

## Enumerator

- SBC\_UJA\_MTPNV\_STAT\_ECCS\_NO** No bit failure detected in non-volatile memory.
- SBC\_UJA\_MTPNV\_STAT\_ECCS** Bit failure detected and corrected in non-volatile memory.

Definition at line 974 of file sbc\_uja116x\_driver.h.

## 16.134.5.19 enum sbc\_mtpnv\_stat\_nvmps\_t

MTPNV status register, non-volatile memory programming status (0x70).

Implements : sbc\_mtpnv\_stat\_nvmps\_t\_Class

## Enumerator

- SBC\_UJA\_MTPNV\_STAT\_NVMPs\_NO** MTPNV memory cannot be overwritten.
- SBC\_UJA\_MTPNV\_STAT\_NVMPs** MTPNV memory is ready to be reprogrammed.

Definition at line 986 of file sbc\_uja116x\_driver.h.



16.134.5.20 enum sbc\_register\_t

Register map.

Implements : sbc\_register\_t\_Class

Enumerator

***SBC\_UJA\_WTDOG\_CTR***  
***SBC\_UJA\_MODE***  
***SBC\_UJA\_FAIL\_SAFE***  
***SBC\_UJA\_MAIN***  
***SBC\_UJA\_SYSTEM\_EVNT***  
***SBC\_UJA\_WTDOG\_STAT***  
***SBC\_UJA\_MEMORY\_0***  
***SBC\_UJA\_MEMORY\_1***  
***SBC\_UJA\_MEMORY\_2***  
***SBC\_UJA\_MEMORY\_3***  
***SBC\_UJA\_LOCK***  
***SBC\_UJA\_REGULATOR***  
***SBC\_UJA\_SUPPLY\_STAT***  
***SBC\_UJA\_SUPPLY\_EVNT***  
***SBC\_UJA\_CAN***  
***SBC\_UJA\_TRANS\_STAT***  
***SBC\_UJA\_TRANS\_EVNT***  
***SBC\_UJA\_DAT\_RATE***  
***SBC\_UJA\_IDENTIF\_0***  
***SBC\_UJA\_IDENTIF\_1***  
***SBC\_UJA\_IDENTIF\_2***  
***SBC\_UJA\_IDENTIF\_3***  
***SBC\_UJA\_MASK\_0***  
***SBC\_UJA\_MASK\_1***  
***SBC\_UJA\_MASK\_2***  
***SBC\_UJA\_MASK\_3***  
***SBC\_UJA\_FRAME\_CTR***  
***SBC\_UJA\_DAT\_MASK\_0***  
***SBC\_UJA\_DAT\_MASK\_1***  
***SBC\_UJA\_DAT\_MASK\_2***  
***SBC\_UJA\_DAT\_MASK\_3***  
***SBC\_UJA\_DAT\_MASK\_4***  
***SBC\_UJA\_DAT\_MASK\_5***  
***SBC\_UJA\_DAT\_MASK\_6***  
***SBC\_UJA\_DAT\_MASK\_7***  
***SBC\_UJA\_WAKE\_STAT***  
***SBC\_UJA\_WAKE\_EN***  
***SBC\_UJA\_GL\_EVNT\_STAT***  
***SBC\_UJA\_SYS\_EVNT\_STAT***  
***SBC\_UJA\_SUP\_EVNT\_STAT***

***SBC\_UJA\_TRANS\_EVNT\_STAT***

***SBC\_UJA\_WAKE\_EVNT\_STAT***

***SBC\_UJA\_MTPNV\_STAT***

***SBC\_UJA\_START\_UP***

***SBC\_UJA\_SBC***

***SBC\_UJA\_MTPNV\_CRC***

***SBC\_UJA\_IDENTIF***

Definition at line 51 of file sbc\_uja116x\_driver.h.

16.134.5.21 enum sbc\_regulator\_pdc\_t

Regulator control register, power distribution control (0x10). PDC is not available on UJA1168 device variants, use any of these two values, the value written to the device will be ignored.

Implements : sbc\_regulator\_pdc\_t\_Class

#### Enumerator

***SBC\_UJA\_REGULATOR\_PDC\_HV*** V1 threshold current for activating the external PNP transistor, load current rising; lth(act)PNP (higher value) V1 threshold current for deactivating the external PNP transistor, load current falling; lth(deact)PNP (higher value).

***SBC\_UJA\_REGULATOR\_PDC\_LV*** V1 threshold current for activating the external PNP transistor; load current rising; lth(act)PNP (lower value) V1 threshold current for deactivating the external PNP transistor; load current falling; lth(deact)PNP (lower value).

Definition at line 344 of file sbc\_uja116x\_driver.h.

16.134.5.22 enum sbc\_regulator\_v1rtc\_t

Regulator control register, set V1 reset threshold (0x10).

Implements : sbc\_regulator\_v1rtc\_t\_Class

#### Enumerator

***SBC\_UJA\_REGULATOR\_V1RTC\_90*** Reset threshold set to 90 % of V1 nominal output voltage.

***SBC\_UJA\_REGULATOR\_V1RTC\_80*** Reset threshold set to 80 % of V1 nominal output voltage.

***SBC\_UJA\_REGULATOR\_V1RTC\_70*** Reset threshold set to 70 % of V1 nominal output voltage.

***SBC\_UJA\_REGULATOR\_V1RTC\_60*** Reset threshold set to 60 % of V1 nominal output voltage.

Definition at line 378 of file sbc\_uja116x\_driver.h.

16.134.5.23 enum sbc\_regulator\_v2c\_t

Regulator control register, V2/VEXT configuration (0x10).

Implements : sbc\_regulator\_v2c\_t\_Class

#### Enumerator

***SBC\_UJA\_REGULATOR\_V2C\_OFF*** V2/VEXT off in all modes.

***SBC\_UJA\_REGULATOR\_V2C\_N*** V2/VEXT on in Normal mode.

***SBC\_UJA\_REGULATOR\_V2C\_N\_S\_R*** V2/VEXT on in Normal, Standby and Reset modes.

***SBC\_UJA\_REGULATOR\_V2C\_N\_S\_S\_R*** V2/VEXT on in Normal, Standby, Sleep and Reset modes.

Definition at line 362 of file sbc\_uja116x\_driver.h.

16.134.5.24 enum **sbc\_sbc\_fnmc\_t**

SBC configuration control register, Forced Normal mode control (0x74).

Implements : **sbc\_sbc\_fnmc\_t\_Class**

## Enumerator

**SBC\_UJA\_SBC\_FNMC\_DIS** Forced Normal mode disabled.

**SBC\_UJA\_SBC\_FNMC\_EN** Forced Normal mode enabled.

Definition at line 1043 of file **sbc\_uja116x\_driver.h**.

16.134.5.25 enum **sbc\_sbc\_sdmc\_t**

SBC configuration control register, Software Development mode control (0x74).

Implements : **sbc\_sbc\_sdmc\_t\_Class**

## Enumerator

**SBC\_UJA\_SBC\_SDMC\_DIS** Software Development mode disabled.

**SBC\_UJA\_SBC\_SDMC\_EN** Software Development mode enabled.

Definition at line 1056 of file **sbc\_uja116x\_driver.h**.

16.134.5.26 enum **sbc\_sbc\_slpc\_t**

SBC configuration control register, Sleep control (0x74).

Implements : **sbc\_sbc\_slpc\_t\_Class**

## Enumerator

**SBC\_UJA\_SBC\_SLPC\_AC** Sleep mode commands accepted. Factory preset value.

**SBC\_UJA\_SBC\_SLPC\_IG** Sleep mode commands ignored.

Definition at line 1069 of file **sbc\_uja116x\_driver.h**.

16.134.5.27 enum **sbc\_sbc\_v1rtsuc\_t**

SBC configuration control register, V1 undervoltage threshold (defined by bit V1RTC) at start-up (0x74).

Implements : **sbc\_sbc\_v1rtsuc\_t\_Class**

## Enumerator

**SBC\_UJA\_SBC\_V1RTSUC\_90** V1 undervoltage detection at 90 % of nominal value at start-up (V1RTC = 00).

**SBC\_UJA\_SBC\_V1RTSUC\_80** V1 undervoltage detection at 80 % of nominal value at start-up (V1RTC = 01).

**SBC\_UJA\_SBC\_V1RTSUC\_70** V1 undervoltage detection at 70 % of nominal value at start-up V1RTC = 10).

**SBC\_UJA\_SBC\_V1RTSUC\_60** V1 undervoltage detection at 60 % of nominal value at start-up (V1RTC = 11).

Definition at line 1027 of file **sbc\_uja116x\_driver.h**.

16.134.5.28 enum **sbc\_start\_up\_rlc\_t**

Start-up control register, RSTN output reset pulse width macros (0x73).

Implements : **sbc\_start\_up\_rlc\_t\_Class**

## Enumerator

**SBC\_UJA\_START\_UP\_RLC\_20\_25p0** Tw(rst) = 20 ms to 25 ms.  
**SBC\_UJA\_START\_UP\_RLC\_10\_12p5** Tw(rst) = 10 ms to 12.5 ms.  
**SBC\_UJA\_START\_UP\_RLC\_03p6\_05** Tw(rst) = 3.6 ms to 5 ms.  
**SBC\_UJA\_START\_UP\_RLC\_01\_01p5** Tw(rst) = 1 ms to 1.5 ms.

Definition at line 998 of file sbc\_uja116x\_driver.h.

16.134.5.29 enum sbc\_start\_up\_v2suc\_t

Start-up control register, V2/VEXT start-up control (0x73).

Implements : sbc\_start\_up\_v2suc\_t\_Class

## Enumerator

**SBC\_UJA\_START\_UP\_V2SUC\_00** bits V2C/VEXTC set to 00 at power-up.  
**SBC\_UJA\_START\_UP\_V2SUC\_11** bits V2C/VEXTC set to 11 at power-up.

Definition at line 1014 of file sbc\_uja116x\_driver.h.

16.134.5.30 enum sbc\_sup\_evnt\_stat\_v1u\_t

Supply event status register, V1 undervoltage (0x62).

Implements : sbc\_sup\_evnt\_stat\_v1u\_t\_Class

## Enumerator

**SBC\_UJA\_SUP\_EVNT\_STAT\_V1U\_NO** no V1 undervoltage event captured.  
**SBC\_UJA\_SUP\_EVNT\_STAT\_V1U** voltage on V1 has dropped below the 90 % undervoltage threshold while V1 is active (event is not captured in Sleep mode because V1 is off); V1U event capture is independent of the setting of bits V1RTC.

Definition at line 876 of file sbc\_uja116x\_driver.h.

16.134.5.31 enum sbc\_sup\_evnt\_stat\_v2o\_t

Supply event status register, V2/VEXT overvoltage (0x62).

Implements : sbc\_sup\_evnt\_stat\_v2o\_t\_Class

## Enumerator

**SBC\_UJA\_SUP\_EVNT\_STAT\_V2O\_NO** No V2/VEXT overvoltage event captured.  
**SBC\_UJA\_SUP\_EVNT\_STAT\_V2O** V2/VEXT overvoltage event captured.

Definition at line 852 of file sbc\_uja116x\_driver.h.

16.134.5.32 enum sbc\_sup\_evnt\_stat\_v2u\_t

Supply event status register, V2/VEXT undervoltage (0x62).

Implements : sbc\_sup\_evnt\_stat\_v2u\_t\_Class

## Enumerator

**SBC\_UJA\_SUP\_EVNT\_STAT\_V2U\_NO** No V2/VEXT undervoltage event captured.  
**SBC\_UJA\_SUP\_EVNT\_STAT\_V2U** V2/VEXT undervoltage event captured.

Definition at line 864 of file sbc\_uja116x\_driver.h.

16.134.5.33 enum `sbc_supply_evnt_v1ue_t`

Supply event capture enable register, V1 undervoltage enable (0x1C).

Implements : `sbc_supply_evnt_v1ue_t_Class`

## Enumerator

**`SBC_UJA_SUPPLY_EVNT_V1UE_DIS`** V1 undervoltage detection disabled.

**`SBC_UJA_SUPPLY_EVNT_V1UE_EN`** V1 undervoltage detection enabled.

Definition at line 448 of file `sbc_uja116x_driver.h`.

16.134.5.34 enum `sbc_supply_evnt_v2oe_t`

Supply event capture enable register, V2/VEXT overvoltage enable (0x1C).

Implements : `sbc_supply_evnt_v2oe_t_Class`

## Enumerator

**`SBC_UJA_SUPPLY_EVNT_V2OE_DIS`** V2/VEXT overvoltage detection disabled.

**`SBC_UJA_SUPPLY_EVNT_V2OE_EN`** V2/VEXT overvoltage detection enabled.

Definition at line 423 of file `sbc_uja116x_driver.h`.

16.134.5.35 enum `sbc_supply_evnt_v2ue_t`

Supply event capture enable register, V2/VEXT undervoltage enable (0x1C).

Implements : `sbc_supply_evnt_v2ue_t_Class`

## Enumerator

**`SBC_UJA_SUPPLY_EVNT_V2UE_DIS`** V2/VEXT undervoltage detection disabled.

**`SBC_UJA_SUPPLY_EVNT_V2UE_EN`** V2/VEXT undervoltage detection enabled.

Definition at line 436 of file `sbc_uja116x_driver.h`.

16.134.5.36 enum `sbc_supply_stat_v1s_t`

Supply voltage status register, V1 status (0x1B).

Implements : `sbc_supply_stat_v1s_t_Class`

## Enumerator

**`SBC_UJA_SUPPLY_STAT_V1S_VAB`** V1 output voltage above 90 % undervoltage threshold.

**`SBC_UJA_SUPPLY_STAT_V1S_VBE`** V1 output voltage below 90 % undervoltage threshold.

Definition at line 410 of file `sbc_uja116x_driver.h`.

16.134.5.37 enum `sbc_supply_stat_v2s_t`

Supply voltage status register, V2/VEXT status (0x1B).

Implements : `sbc_supply_stat_v2s_t_Class`

## Enumerator

**`SBC_UJA_SUPPLY_STAT_V2S_VOK`** V2/VEXT voltage ok.

**`SBC_UJA_SUPPLY_STAT_V2S_VBE`** V2/VEXT output voltage below undervoltage threshold

**`SBC_UJA_SUPPLY_STAT_V2S_VAB`** V2/VEXT output voltage above overvoltage threshold

**`SBC_UJA_SUPPLY_STAT_V2S_DIS`** V2/VEXT disabled

Definition at line 394 of file `sbc_uja116x_driver.h`.

**16.134.5.38 enum sbc\_sys\_evnt\_otwe\_t**

System event capture enable, overtemperature warning enable (0x04).

Implements : sbc\_sys\_evnt\_otwe\_t\_Class

**Enumerator**

**SBC\_UJA\_SYS\_EVNT\_OTWE\_DIS** Overtemperature warning disabled.

**SBC\_UJA\_SYS\_EVNT\_OTWE\_EN** Overtemperature warning enabled.

Definition at line 251 of file sbc\_uja116x\_driver.h.

**16.134.5.39 enum sbc\_sys\_evnt\_spife\_t**

System event capture enable, SPI failure enable (0x04).

Implements : sbc\_sys\_evnt\_spife\_t\_Class

**Enumerator**

**SBC\_UJA\_SYS\_EVNT\_SPIFE\_DIS** SPI failure detection disabled.

**SBC\_UJA\_SYS\_EVNT\_SPIFE\_EN** SPI failure detection enabled.

Definition at line 263 of file sbc\_uja116x\_driver.h.

**16.134.5.40 enum sbc\_sys\_evnt\_stat\_otw\_t**

System event status register, overtemperature warning (0x61).

Implements : sbc\_sys\_evnt\_stat\_otw\_t\_Class

**Enumerator**

**SBC\_UJA\_SYS\_EVNT\_STAT\_OTW\_NO** Overtemperature not detected.

**SBC\_UJA\_SYS\_EVNT\_STAT\_OTW** The global chip temperature has exceeded the overtemperature warning threshold, Tth(warn)otp (not in Sleep mode).

Definition at line 809 of file sbc\_uja116x\_driver.h.

**16.134.5.41 enum sbc\_sys\_evnt\_stat\_po\_t**

System event status register, power-on (0x61).

Implements : sbc\_sys\_evnt\_stat\_po\_t\_Class

**Enumerator**

**SBC\_UJA\_SYS\_EVNT\_STAT\_PO\_NO** No recent battery power-on.

**SBC\_UJA\_SYS\_EVNT\_STAT\_PO** The UJA116xA has left Off mode after battery power-on.

Definition at line 797 of file sbc\_uja116x\_driver.h.

**16.134.5.42 enum sbc\_sys\_evnt\_stat\_spif\_t**

System event status register, SPI failure (0x61).

Implements : sbc\_sys\_evnt\_stat\_spif\_t\_Class

**Enumerator**

**SBC\_UJA\_SYS\_EVNT\_STAT\_SPIF\_NO** No SPI failure detected

**SBC\_UJA\_SYS\_EVNT\_STAT\_SPIF** SPI clock count error (only 16-, 24- and 32-bit commands are valid), illegal WMC, NWP or MC code or attempted write access to locked register (not in Sleep mode)

Definition at line 822 of file sbc\_uja116x\_driver.h.

16.134.5.43 enum `sbc_sys_evnt_stat_wdf_t`

System event status register, watchdog failure (0x61).

Implements : `sbc_sys_evnt_stat_wdf_t_Class`

## Enumerator

**`SBC_UJA_SYS_EVNT_STAT_WDF_NO`** No watchdog failure event captured

**`SBC_UJA_SYS_EVNT_STAT_WDF`** Watchdog overflow in Window or Timeout mode or watchdog triggered too early in Window mode; a system reset is triggered immediately in response to a watchdog failure in Window mode; when the watchdog overflows in Timeout mode, a system reset is only performed if a WDF is already pending (WDF = 1).

Definition at line 836 of file `sbc_uja116x_driver.h`.

16.134.5.44 enum `sbc_trans_evnt_cbse_t`

Transceiver event capture enable register, CAN-bus silence enable (0x23).

Implements : `sbc_trans_evnt_cbse_t_Class`

## Enumerator

**`SBC_UJA_TRANS_EVNT_CBSE_DIS`** CAN-bus silence detection disabled.

**`SBC_UJA_TRANS_EVNT_CBSE_EN`** CAN-bus silence detection enabled.

Definition at line 597 of file `sbc_uja116x_driver.h`.

16.134.5.45 enum `sbc_trans_evnt_cfe_t`

Transceiver event capture enable register, CAN failure enable (0x23).

Implements : `sbc_trans_evnt_cfe_t_Class`

## Enumerator

**`SBC_UJA_TRANS_EVNT_CFE_DIS`** CAN failure detection disabled.

**`SBC_UJA_TRANS_EVNT_CFE_EN`** CAN failure detection enabled.

Definition at line 609 of file `sbc_uja116x_driver.h`.

16.134.5.46 enum `sbc_trans_evnt_cwe_t`

Transceiver event capture enable register, CAN wake-up enable (0x23).

Implements : `sbc_trans_evnt_cwe_t_Class`

## Enumerator

**`SBC_UJA_TRANS_EVNT_CWE_DIS`** CAN wake-up detection disabled.

**`SBC_UJA_TRANS_EVNT_CWE_EN`** CAN wake-up detection enabled.

Definition at line 621 of file `sbc_uja116x_driver.h`.

16.134.5.47 enum `sbc_trans_evnt_stat_cbs_t`

Transceiver event status register, CAN-bus status (0x63).

Implements : `sbc_trans_evnt_stat_cbs_t_Class`

## Enumerator

**`SBC_UJA_TRANS_EVNT_STAT_CBS_NO`** CAN-bus active.

**SBC\_UJA\_TRANS\_EVT\_STAT\_CBS** No activity on CAN-bus for tto(silence) (detected only when CBSE = 1 while bus active).

Definition at line 903 of file sbc\_uja116x\_driver.h.

16.134.5.48 enum sbc\_trans\_evt\_stat\_cf\_t

Transceiver event status register, CAN failure (0x63).

Implements : sbc\_trans\_evt\_stat\_cf\_t\_Class

#### Enumerator

**SBC\_UJA\_TRANS\_EVT\_STAT\_CF\_NO** No CAN failure detected.

**SBC\_UJA\_TRANS\_EVT\_STAT\_CF** CAN transceiver deactivated due to VCAN undervoltage OR dominant clamped TXD (not in Sleep mode)

Definition at line 916 of file sbc\_uja116x\_driver.h.

16.134.5.49 enum sbc\_trans\_evt\_stat\_cw\_t

Transceiver event status register, CAN wake-up (0x63).

Implements : sbc\_trans\_evt\_stat\_cw\_t\_Class

#### Enumerator

**SBC\_UJA\_TRANS\_EVT\_STAT\_CW\_NO** No CAN wake-up event detected.

**SBC\_UJA\_TRANS\_EVT\_STAT\_CW** CAN wake-up event detected while the transceiver is in CAN Offline Mode.

Definition at line 929 of file sbc\_uja116x\_driver.h.

16.134.5.50 enum sbc\_trans\_evt\_stat\_pnfde\_t

Transceiver event status register, partial networking frame detection error (0x63).

Implements : sbc\_trans\_evt\_stat\_pnfde\_t\_Class

#### Enumerator

**SBC\_UJA\_TRANS\_EVT\_STAT\_PNFDE\_NO** No partial networking frame detection error detected.

**SBC\_UJA\_TRANS\_EVT\_STAT\_PNFDE** Partial networking frame detection error detected.

Definition at line 891 of file sbc\_uja116x\_driver.h.

16.134.5.51 enum sbc\_trans\_stat\_cbss\_t

Transceiver status register, CAN-bus silence status (0x22).

Implements : sbc\_trans\_stat\_cbss\_t\_Class

#### Enumerator

**SBC\_UJA\_TRANS\_STAT\_CBSS\_ACT** CAN-bus active (communication detected on bus)

**SBC\_UJA\_TRANS\_STAT\_CBSS\_INACT** CAN-bus inactive (for longer than t\_to(silence)).

Definition at line 561 of file sbc\_uja116x\_driver.h.

16.134.5.52 enum sbc\_trans\_stat\_cfs\_t

Transceiver status register, CAN failure status (0x22).

Implements : sbc\_trans\_stat\_cfs\_t\_Class



## Enumerator

**SBC\_UJA\_TRANS\_STAT\_CFS\_NO\_TXD** No TXD dominant time-out event detected.

**SBC\_UJA\_TRANS\_STAT\_CFS\_TXD** CAN transmitter disabled due to a TXD dominant time-out event.

Definition at line 585 of file sbc\_uja116x\_driver.h.

## 16.134.5.53 enum sbc\_trans\_stat\_coscs\_t

Transceiver status register, CAN oscillator status (0x22).

Implements : sbc\_trans\_stat\_coscs\_t\_Class

## Enumerator

**SBC\_UJA\_TRANS\_STAT\_COSCS\_NRUN** CAN partial networking oscillator not running at target frequency.

**SBC\_UJA\_TRANS\_STAT\_COSCS\_RUN** CAN partial networking oscillator running at target.

Definition at line 549 of file sbc\_uja116x\_driver.h.

## 16.134.5.54 enum sbc\_trans\_stat\_cpnerr\_t

Transceiver status register, CAN partial networking error (0x22).

Implements : sbc\_trans\_stat\_cpnerr\_t\_Class

## Enumerator

**SBC\_UJA\_TRANS\_STAT\_CPNERR\_NO\_DET** no CAN partial networking error detected (PNFDE = 0 AND PNCOK = 1).

**SBC\_UJA\_TRANS\_STAT\_CPNERR\_DET** CAN partial networking error detected (PNFDE = 1 OR PNCOK = 0; wake-up via standard wake-up pattern only).

Definition at line 524 of file sbc\_uja116x\_driver.h.

## 16.134.5.55 enum sbc\_trans\_stat\_cpns\_t

Transceiver status register, CAN partial networking status (0x22).

Implements : sbc\_trans\_stat\_cpns\_t\_Class

## Enumerator

**SBC\_UJA\_TRANS\_STAT\_CPNS\_ERR** CAN partial networking configuration error detected (PNCOK = 0).

**SBC\_UJA\_TRANS\_STAT\_CPNS\_OK** CAN partial networking configuration ok (PNCOK = 1).

Definition at line 537 of file sbc\_uja116x\_driver.h.

## 16.134.5.56 enum sbc\_trans\_stat\_cts\_t

Transceiver status register, CAN transceiver status (0x22).

Implements : sbc\_trans\_stat\_cts\_t\_Class

## Enumerator

**SBC\_UJA\_TRANS\_STAT\_CTS\_INACT** CAN transceiver not in Active mode.

**SBC\_UJA\_TRANS\_STAT\_CTS\_ACT** CAN transceiver in Active mode.

Definition at line 512 of file sbc\_uja116x\_driver.h.

#### 16.134.5.57 enum sbc\_trans\_stat\_vcs\_t

Transceiver status register, VCAN status (0x22).

Implements : sbc\_trans\_stat\_vcs\_t\_Class

##### Enumerator

**SBC\_UJA\_TRANS\_STAT\_VCS\_AB** CAN supply voltage is above the 90 % threshold.

**SBC\_UJA\_TRANS\_STAT\_VCS\_BE** CAN supply voltage is below the 90 % threshold

Definition at line 573 of file sbc\_uja116x\_driver.h.

#### 16.134.5.58 enum sbc\_wake\_en\_wpfe\_t

WAKE pin event capture enable register, WAKE pin falling-edge enable (0x4C).

Implements : sbc\_wake\_en\_wpfe\_t\_Class

##### Enumerator

**SBC\_UJA\_WAKE\_EN\_WPFE\_DIS** Falling-edge detection on WAKE pin disabled.

**SBC\_UJA\_WAKE\_EN\_WPFE\_EN** Falling-edge detection on WAKE pin enabled.

Definition at line 737 of file sbc\_uja116x\_driver.h.

#### 16.134.5.59 enum sbc\_wake\_en\_wpre\_t

WAKE pin event capture enable register, WAKE pin rising-edge enable (0x4C).

Implements : sbc\_wake\_en\_wpre\_t\_Class

##### Enumerator

**SBC\_UJA\_WAKE\_EN\_WPRE\_DIS** Rising-edge detection on WAKE pin disabled.

**SBC\_UJA\_WAKE\_EN\_WPRE\_EN** Rising-edge detection on WAKE pin enabled.

Definition at line 725 of file sbc\_uja116x\_driver.h.

#### 16.134.5.60 enum sbc\_wake\_evnt\_stat\_wpf\_t

WAKE pin event status register, WAKE pin falling edge (0x64).

Implements : sbc\_wake\_evnt\_stat\_wpf\_t\_Class

##### Enumerator

**SBC\_UJA\_WAKE\_EVNT\_STAT\_WPF\_NO** No falling edge detected on WAKE pin.

**SBC\_UJA\_WAKE\_EVNT\_STAT\_WPF** Falling edge detected on WAKE pin.

Definition at line 953 of file sbc\_uja116x\_driver.h.

#### 16.134.5.61 enum sbc\_wake\_evnt\_stat\_wpr\_t

WAKE pin event status register, WAKE pin rising edge (0x64).

Implements : sbc\_wake\_evnt\_stat\_wpr\_t\_Class

##### Enumerator

**SBC\_UJA\_WAKE\_EVNT\_STAT\_WPR\_NO** No rising edge detected on WAKE pin.

**SBC\_UJA\_WAKE\_EVNT\_STAT\_WPR** Rising edge detected on WAKE pin.

Definition at line 941 of file sbc\_uja116x\_driver.h.

16.134.5.62 enum `sbc_wake_stat_wpvs_t`

WAKE pin status register, WAKE pin status (0x4B).

Implements : `sbc_wake_stat_wpvs_t_Class`

## Enumerator

**`SBC_UJA_WAKE_STAT_WPVS_BE`** Voltage on WAKE pin below switching threshold ( $V_{th}(sw)$ ).

**`SBC_UJA_WAKE_STAT_WPVS_AB`** voltage on WAKE pin above switching threshold ( $V_{th}(sw)$ ).

Definition at line 713 of file `sbc_uja116x_driver.h`.

16.134.5.63 enum `sbc_wtdog_ctr_nwp_t`

Watchdog control register, nominal watchdog period (0x00). Eight watchdog periods are supported, from 8 ms to 4096 ms. The watchdog period is programmed via bits NWP. The selected period is valid for both Window and Timeout modes. The default watchdog period is 128 ms. A watchdog trigger event resets the watchdog timer. A watchdog trigger event is any valid write access to the Watchdog control register. If the watchdog mode or the watchdog period have changed as a result of the write access, the new values are immediately valid.

Implements : `sbc_wtdog_ctr_nwp_t_Class`

## Enumerator

**`SBC_UJA_WTD OG_CTR_NWP_8`** 8 ms.

**`SBC_UJA_WTD OG_CTR_NWP_16`** 16 ms.

**`SBC_UJA_WTD OG_CTR_NWP_32`** 32 ms.

**`SBC_UJA_WTD OG_CTR_NWP_64`** 64 ms.

**`SBC_UJA_WTD OG_CTR_NWP_128`** 128 ms.

**`SBC_UJA_WTD OG_CTR_NWP_256`** 256 ms.

**`SBC_UJA_WTD OG_CTR_NWP_1024`** 1024 ms.

**`SBC_UJA_WTD OG_CTR_NWP_4096`** 4096 ms.

Definition at line 149 of file `sbc_uja116x_driver.h`.

16.134.5.64 enum `sbc_wtdog_ctr_wmc_t`

Watchdog control register, watchdog mode control (0x00). The UJA116xA contains a watchdog that supports three operating modes: Window, Timeout and Autonomous. In Window mode (available only in SBC Normal mode), a watchdog trigger event within a defined watchdog window triggers and resets the watchdog timer. In Timeout mode, the watchdog runs continuously and can be triggered and reset at any time within the watchdog period by a watchdog trigger. Watchdog time-out mode can also be used for cyclic wake-up of the microcontroller. In Autonomous mode, the watchdog can be off or autonomously in Timeout mode, depending on the selected SBC mode. The watchdog mode is selected via bits WMC in the Watchdog control register. The SBC must be in Standby mode when the watchdog mode is changed.

Implements : `sbc_wtdog_ctr_wmc_t_Class`

## Enumerator

**`SBC_UJA_WTD OG_CTR_WMC_AUTO`** Autonomous mode.

**`SBC_UJA_WTD OG_CTR_WMC_TIME`** Timeout mode.

**`SBC_UJA_WTD OG_CTR_WMC_WIND`** Window mode (available only in SBC Normal mode).

Definition at line 128 of file `sbc_uja116x_driver.h`.

**16.134.5.65 enum sbc\_wtdog\_stat\_fnms\_t**

Watchdog status register, forced Normal mode status (0x05).

Implements : sbc\_wtdog\_stat\_fnms\_t\_Class

**Enumerator**

***SBC\_UJA\_WTD OG\_STAT\_FNMS\_N\_NORMAL*** SBC is not in Forced Normal mode.

***SBC\_UJA\_WTD OG\_STAT\_FNMS\_NORMAL*** SBC is in Forced Normal mode.

Definition at line 275 of file sbc\_uja116x\_driver.h.

**16.134.5.66 enum sbc\_wtdog\_stat\_sdms\_t**

Watchdog status register, Software Development mode status (0x05).

Implements : sbc\_wtdog\_stat\_sdms\_t\_Class

**Enumerator**

***SBC\_UJA\_WTD OG\_STAT\_SDMS\_N\_NORMAL*** SBC is not in Software Development mode.

***SBC\_UJA\_WTD OG\_STAT\_SDMS\_NORMAL*** SBC is in Software Development mode.

Definition at line 287 of file sbc\_uja116x\_driver.h.

**16.134.5.67 enum sbc\_wtdog\_stat\_wds\_t**

Watchdog status register, watchdog status (0x05).

Implements : sbc\_wtdog\_stat\_wds\_t\_Class

**Enumerator**

***SBC\_UJA\_WTD OG\_STAT\_WDS\_OFF*** Watchdog is off.

***SBC\_UJA\_WTD OG\_STAT\_WDS\_FIH*** Watchdog is in first half of the nominal period.

***SBC\_UJA\_WTD OG\_STAT\_WDS\_SEH*** Watchdog is in second half of the nominal period.

Definition at line 299 of file sbc\_uja116x\_driver.h.

## 16.135 Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL)

### 16.135.1 Detailed Description

The S32 SDK provides a Peripheral Abstraction Layer for Universal Asynchronous Receiver-Transmitter (UART) modules of S32 SDK devices.

The UART PAL driver allows communication over a serial port. It was designed to be portable across all platforms and IPs which support UART communication.

#### How to integrate UART PAL in your application

Unlike the other drivers, UART PAL modules need to include a configuration file named `uart_pal_cfg.h`, which allows the user to specify which IPs are used and how many resources are allocated for each of them (state structures). The following code example shows how to configure one instance for each available UART IPs.

```
#ifndef uart_pal_cfg_H
#define uart_pal_cfg_H

/* Define which IP instance will be used in current project */
#define UART_OVER_LPUART
#define UART_OVER_FLEXIO
#define UART_OVER_LINFLEXD

/* Define the resources necessary for current project */
#define NO_OF_LPUART_INSTS_FOR_UART 1U
#define NO_OF_FLEXIO_INSTS_FOR_UART 1U
#define NO_OF_LINFLEXD_INSTS_FOR_UART 1U

#endif /* uart_pal_cfg_H */
```

The following table contains the matching between platforms and available IPs

IP	S32-K11	S32-K11	S32-K14	S32-K14	S32-K14	S32-K14	S32-V23	MPC577G	MPC577C	MPC577P	S32-R27	S32-R37	MPC577R	MPC577C	S32-R29	S32-G27A	S32-R45	S32-K144
LPURT	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	Yes
FLEXIO	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No	No	No	Yes

L← I← N← Flex← D← _← _← U← A← R← T	N← O	N← O	N← O	N← O	N← O	N← O	Y← E← S	Y← E← S	Y← E← S	Y← E← S	Y← E← S	Y← E← S	Y← E← S	N← O	Y← E← S	Y← E← S	Y← E← S	N← O
e← S← C← I← _← _← U← A← R← T	N← O	N← O	N← O	N← O	N← O	N← O	N← O	N← O	N← O	N← O	N← O	N← O	N← O	Y← E← S	N← O	N← O	N← O	N← O

### Features

- Interrupt or DMA mode
- Provides blocking and non-blocking transmit and receive functions
- Configurable baud rate and number of bits per char

The following table contains the matching between IPs and available features

IP/FEATURE	Bits per char	Parity	Stop Bits
LPUART	8, 9, 10	Disabled, Even, Odd	1, 2
FLEXIO_UART	7, 8, 9, 10, 15, 16	Disabled	1
LINFlexD_UART	7, 8, 15, 16	Disabled, Even, Odd	1, 2

### Functionality

#### Initialization

In order to use the UART PAL driver it must be first initialized, using [UART\\_Init\(\)](#) function. Once initialized, it cannot be initialized again for the same UART module instance until it is de-initialized, using [UART\\_Deinit\(\)](#). The initialization function does the following operations:

- sets the baud rate
- sets parity/bit count/stop bits count
- initializes the state structure for the current instance
- enables receiver/transmitter for the current instance Different UART modules instances can function independently of each other.

#### Interrupt-based communication

After initialization, a serial communication can be triggered by calling [UART\\_SendData](#) function. The driver interrupt handler takes care of transmitting all bytes in the TX buffer. Similarly, data reception is triggered by calling [UART\\_ReceiveData](#) function, passing the RX buffer as parameter. The driver interrupt handler reads the received byte

and saves them in the RX buffer. Non-blocking operations will initiate the transfer and return `STATUS_SUCCESS`, but the module is still busy with the transfer and another transfer can't be initiated until the current transfer is complete. The application can check the status of the current transfer by calling `UART_GetTransmitStatus()` / `UART_GetReceiveStatus()`.

The workflow applies to send/receive operations using blocking method (triggered by `UART_SendDataBlocking()` and `UART_ReceiveDataBlocking()`), with the single difference that the send/receive function will not return until the send/receive operation is complete (all bytes are successfully transferred or a timeout occurred). The timeout for the blocking method is passed as parameter by the user.

When configured to use the LPUART or LINFlexD peripherals, if a user callback is installed for RX/TX, the callback has to take care of data handling and aborting the transfer when complete; the driver interrupt handler does not manipulate the buffers in this case. When using the UART PAL over FLEXIO, when the driver completes the transmission or reception of the current buffer, it will invoke the user callback (if installed) with an appropriate event.

#### DMA-based communication

In DMA operation, both blocking and non-blocking transmission methods configure a DMA channel to copy data to/from the buffer. The driver assumes the DMA channel is already allocated. In case of LPUART and LINFlexD, the application also assumes that the proper requests are routed to it via DMAMUX. The FLEXIO driver will set the DMA request source. After configuring the DMA channel, the driver enables DMA requests for RX/TX, then the DMA engine takes care of moving data to/from the data buffer. In this scenario, the callback is only called when the full transmission is done, that is when the DMA channel finishes the number of loops configured in the transfer descriptor.

#### Important Notes

- Before using the UART PAL driver the module clock must be configured. Refer to Clock Manager for clock configuration.
- The driver enables the interrupts for the corresponding UART module, but any interrupt priority must be done by the application
- The board specific configurations must be done prior to driver calls; the driver has no influence on the functionality of the TX/RX pins - they must be configured by application
- DMA module has to be initialized prior to UART usage in DMA mode; also, DMA channels need to be allocated for UART usage by the application (the driver only takes care of configuring the DMA channels received in the configuration structure)
- Some features are not available for all UART IPs and incorrect parameters will be handled by `DEV_ASSERT`
- The `UART_SetBaudRate()` function attempts to configure the requested baud rate for the selected UART peripheral. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences. The application should call `UART_GetBaudRate()` after `UART_SetBaudRate()` to check what baud rate was actually set.
- Due to different implementation of drivers, callback parameters in case of errors during reception may be different. LPUART and LINFLEXD\_UART will pass `UART_EVENT_ERROR` as the event parameter for callbacks, whereas FLEXIO\_UART will pass `UART_EVENT_END_TRANSFER`. In both cases, in order to retrieve the exact status of the latest reception, users can call the appropriate functions in the (`UART_GetReceiveStatus()`).
- Due to DMA mechanism, bit character length should be recommended multiple by 8. So, LPUART only supports 8 bit chars while FLEXIO\_UART supports 8 and 16 bit chars in DMA module. Specially, LINFLEXD\_UART can process with all character lengths currently because it takes care of parity bit handling internally.

#### Integration guideline

##### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\pal\src\uart\uart_pal.c
```

### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\drivers\inc
${S32SDK_PATH}\platform\pal\inc
${S32SDK_PATH}\rtos\osif
```

### Preprocessor symbols

No special symbols are required for this component

### Dependencies

[Clock Manager Interrupt Manager \(Interrupt\) Enhanced Direct Memory Access \(eDMA\) OS Interface \(OSIF\)](#)

### Example code

```
uint32_t bytesRemaining;

/* Instance information structure */
uart_instance_t uart_pall_instance = {
    .instType = UART_INST_TYPE_FLEXIO_UART,
    .instIdx = 0U
};

/* Configure UART */
uart_user_config_t uart_pall_Config0 = {
    .baudRate      = 600U,
    .bitCount      = UART_7_BITS_PER_CHAR,
    .parityMode     = UART_PARITY_DISABLED,
    .stopBitCount  = UART_ONE_STOP_BIT,
    .transferType   = UART_USING_INTERRUPTS,
    .rxDMAChannel  = 0U,
    .txDMAChannel  = 0U,
    .rxCallback     = NULL,
    .rxCallbackParam = NULL,
    .txCallback     = NULL,
    .txCallbackParam = NULL,
    .extension      = NULL
};

/* Configure FLEXIO pins routing */
extension_flexio_for_uart_t extension = {
    .dataPinTx = 0U,
    .dataPinRx = 1U,
};
uart_pall_Config0.extension = &extension;

/* Buffers */
uint8_t tx[8] = {0, 1, 2, 3, 4, 5, 6, 7};
uint8_t rx[8];

/* Initialize UART */
UART_Init(&uart_pall_instance, &uart_pall_Config0);

/* Send 8 frames */
UART_SendData(&uart_pall_instance, tx, 8U);
while(UART_GetTransmitStatus(&uart_pall_instance, &bytesRemaining) !=
    STATUS_SUCCESS);

/* Receive 8 frames */
UART_ReceiveData(&uart_pall_instance, rx, 8UL);
/* Wait for transfer to be completed */
while(UART_GetReceiveStatus(&uart_pall_instance, &bytesRemaining) != STATUS_SUCCESS)
;

/* De-initialize UART */
UART_Deinit(&uart_pall_instance);
```

### Data Structures

- struct `uart_user_config_t`  
Defines the UART configuration structure. [More...](#)



- struct [extension\\_flexio\\_for\\_uart\\_t](#)

*Defines the extension structure for the UART over FLEXIO. [More...](#)*

## Enumerations

- enum [uart\\_bit\\_count\\_per\\_char\\_t](#) {  
[UART\\_7\\_BITS\\_PER\\_CHAR](#) = 0x0U, [UART\\_8\\_BITS\\_PER\\_CHAR](#) = 0x1U, [UART\\_9\\_BITS\\_PER\\_CHAR](#) = 0x2U, [UART\\_10\\_BITS\\_PER\\_CHAR](#) = 0x3U,  
[UART\\_15\\_BITS\\_PER\\_CHAR](#) = 0x4U, [UART\\_16\\_BITS\\_PER\\_CHAR](#) = 0x5U }  
*Defines the number of bits in a character.*
- enum [uart\\_transfer\\_type\\_t](#) { [UART\\_USING\\_DMA](#) = 0U, [UART\\_USING\\_INTERRUPTS](#) = 1U }  
*Defines the transfer type.*
- enum [uart\\_parity\\_mode\\_t](#) { [UART\\_PARITY\\_DISABLED](#) = 0x0U, [UART\\_PARITY\\_EVEN](#) = 0x2U, [UART\\_PARITY\\_ODD](#) = 0x3U }  
*Defines the parity mode.*
- enum [uart\\_stop\\_bit\\_count\\_t](#) { [UART\\_ONE\\_STOP\\_BIT](#) = 0x0U, [UART\\_TWO\\_STOP\\_BIT](#) = 0x1U }  
*Defines the number of stop bits.*

## Functions

- void [UART\\_GetDefaultConfig](#) ([uart\\_user\\_config\\_t](#) \*config)  
*Gets the default configuration structure.*
- status\_t [UART\\_Init](#) (const [uart\\_instance\\_t](#) \*const instance, const [uart\\_user\\_config\\_t](#) \*config)  
*Initializes the UART module.*
- status\_t [UART\\_Deinit](#) (const [uart\\_instance\\_t](#) \*const instance)  
*De-initializes the UART module.*
- status\_t [UART\\_SetBaudRate](#) (const [uart\\_instance\\_t](#) \*const instance, uint32\_t desiredBaudRate)  
*Configures the UART baud rate.*
- status\_t [UART\\_GetBaudRate](#) (const [uart\\_instance\\_t](#) \*const instance, uint32\_t \*configuredBaudRate)  
*Returns the UART baud rate.*
- status\_t [UART\\_SendDataBlocking](#) (const [uart\\_instance\\_t](#) \*const instance, const uint8\_t \*txBuff, uint32\_t txSize, uint32\_t timeout)  
*Perform a blocking UART transmission.*
- status\_t [UART\\_SendData](#) (const [uart\\_instance\\_t](#) \*const instance, const uint8\_t \*txBuff, uint32\_t txSize)  
*Perform a non-blocking UART transmission.*
- status\_t [UART\\_AbortSendingData](#) (const [uart\\_instance\\_t](#) \*const instance)  
*Terminates a non-blocking transmission early.*
- status\_t [UART\\_GetTransmitStatus](#) (const [uart\\_instance\\_t](#) \*const instance, uint32\_t \*bytesRemaining)  
*Get the status of the current non-blocking UART transmission.*
- status\_t [UART\\_ReceiveDataBlocking](#) (const [uart\\_instance\\_t](#) \*const instance, uint8\_t \*rxBuff, uint32\_t rxSize, uint32\_t timeout)  
*Perform a blocking UART reception.*
- status\_t [UART\\_ReceiveData](#) (const [uart\\_instance\\_t](#) \*const instance, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Perform a non-blocking UART reception.*
- status\_t [UART\\_AbortReceivingData](#) (const [uart\\_instance\\_t](#) \*const instance)  
*Terminates a non-blocking receive early.*
- status\_t [UART\\_GetReceiveStatus](#) (const [uart\\_instance\\_t](#) \*const instance, uint32\_t \*bytesRemaining)  
*Get the status of the current non-blocking UART reception.*
- status\_t [UART\\_SetRxBuffer](#) (const [uart\\_instance\\_t](#) \*const instance, uint8\_t \*rxBuff, uint32\_t rxSize)  
*Provide a buffer for receiving data.*
- status\_t [UART\\_SetTxBuffer](#) (const [uart\\_instance\\_t](#) \*const instance, const uint8\_t \*txBuff, uint32\_t txSize)  
*Provide a buffer for transmitting data.*

### 16.135.2 Data Structure Documentation

#### 16.135.2.1 struct uart\_user\_config\_t

Defines the UART configuration structure.

Implements : `uart_user_config_t_Class`

Definition at line 88 of file `uart_pal.h`.

##### Data Fields

- `uint32_t baudRate`
- `uart_bit_count_per_char_t bitCount`
- `uart_parity_mode_t parityMode`
- `uart_stop_bit_count_t stopBitCount`
- `uart_transfer_type_t transferType`
- `uint8_t rxDMAChannel`
- `uint8_t txDMAChannel`
- `uart_callback_t rxCallback`
- `void * rxCallbackParam`
- `uart_callback_t txCallback`
- `void * txCallbackParam`
- `void * extension`

##### Field Documentation

#### 16.135.2.1.1 uint32\_t baudRate

Baud rate

Definition at line 90 of file `uart_pal.h`.

#### 16.135.2.1.2 uart\_bit\_count\_per\_char\_t bitCount

Number of bits in a character

Definition at line 91 of file `uart_pal.h`.

#### 16.135.2.1.3 void\* extension

This field will be used to add extra settings to the basic configuration like FlexIO data pins

Definition at line 101 of file `uart_pal.h`.

#### 16.135.2.1.4 uart\_parity\_mode\_t parityMode

Parity mode, disabled (default), even, odd

Definition at line 92 of file `uart_pal.h`.

#### 16.135.2.1.5 uart\_callback\_t rxCallback

Callback to invoke for data receive

Definition at line 97 of file `uart_pal.h`.

#### 16.135.2.1.6 void\* rxCallbackParam

Receive callback parameter

Definition at line 98 of file `uart_pal.h`.

**16.135.2.1.7 uint8\_t rxDMAChannel**

Channel number for DMA rx channel.

Definition at line 95 of file uart\_pal.h.

**16.135.2.1.8 uart\_stop\_bit\_count\_t stopBitCount**

number of stop bits, 1 stop bit (default) or 2 stop bits

Definition at line 93 of file uart\_pal.h.

**16.135.2.1.9 uart\_transfer\_type\_t transferType**

Type of the transfer (interrupt/dma based)

Definition at line 94 of file uart\_pal.h.

**16.135.2.1.10 uart\_callback\_t txCallback**

Callback to invoke for data send

Definition at line 99 of file uart\_pal.h.

**16.135.2.1.11 void\* txCallbackParam**

Transmit callback parameter

Definition at line 100 of file uart\_pal.h.

**16.135.2.1.12 uint8\_t txDMAChannel**

Channel number for DMA tx channel.

Definition at line 96 of file uart\_pal.h.

**16.135.2.2 struct extension\_flexio\_for\_uart\_t**

Defines the extension structure for the UART over FLEXIO.

Implements : extension\_flexio\_for\_uart\_t\_Class

Definition at line 110 of file uart\_pal.h.

**Data Fields**

- uint8\_t [dataPinTx](#)
- uint8\_t [dataPinRx](#)

**Field Documentation****16.135.2.2.1 uint8\_t dataPinRx**

Flexio pin to use as Rx pin

Definition at line 113 of file uart\_pal.h.

**16.135.2.2.2 uint8\_t dataPinTx**

Flexio pin to use as Tx pin

Definition at line 112 of file uart\_pal.h.

**16.135.3 Enumeration Type Documentation**

### 16.135.3.1 enum `uart_bit_count_per_char_t`

Defines the number of bits in a character.

Implements : `uart_bit_count_per_char_t_Class`

#### Enumerator

**`UART_7_BITS_PER_CHAR`** 7-bit data characters  
**`UART_8_BITS_PER_CHAR`** 8-bit data characters  
**`UART_9_BITS_PER_CHAR`** 9-bit data characters  
**`UART_10_BITS_PER_CHAR`** 10-bit data characters  
**`UART_15_BITS_PER_CHAR`** 15-bit data characters  
**`UART_16_BITS_PER_CHAR`** 16-bit data characters

Definition at line 39 of file `uart_pal.h`.

### 16.135.3.2 enum `uart_parity_mode_t`

Defines the parity mode.

Implements : `uart_parity_mode_t_Class`

#### Enumerator

**`UART_PARITY_DISABLED`** parity disabled  
**`UART_PARITY_EVEN`** parity enabled, type even  
**`UART_PARITY_ODD`** parity enabled, type odd

Definition at line 65 of file `uart_pal.h`.

### 16.135.3.3 enum `uart_stop_bit_count_t`

Defines the number of stop bits.

Implements : `uart_stop_bit_count_t_Class`

#### Enumerator

**`UART_ONE_STOP_BIT`** one stop bit  
**`UART_TWO_STOP_BIT`** two stop bits

Definition at line 77 of file `uart_pal.h`.

### 16.135.3.4 enum `uart_transfer_type_t`

Defines the transfer type.

Implements : `uart_transfer_type_t_Class`

#### Enumerator

**`UART_USING_DMA`** Driver uses DMA for data transfers  
**`UART_USING_INTERRUPTS`** Driver uses interrupts for data transfers

Definition at line 54 of file `uart_pal.h`.

## 16.135.4 Function Documentation

### 16.135.4.1 `status_t UART_AbortReceivingData ( const uart_instance_t *const instance )`

Terminates a non-blocking receive early.

**Parameters**

<i>in</i>	<i>instance</i>	Pointer to the UART_PAL instance structure.
-----------	-----------------	---

**Returns**

STATUS\_SUCCESS: if successful; STATUS\_ERROR : if invalid instance type;

Definition at line 1018 of file uart\_pal.c.

#### 16.135.4.2 status\_t UART\_AbortSendingData ( const uart\_instance\_t \*const *instance* )

Terminates a non-blocking transmission early.

**Parameters**

<i>in</i>	<i>instance</i>	Pointer to the UART_PAL instance structure.
-----------	-----------------	---

**Returns**

STATUS\_SUCCESS: if successful; STATUS\_ERROR : if invalid instance type;

Definition at line 832 of file uart\_pal.c.

#### 16.135.4.3 status\_t UART\_Deinit ( const uart\_instance\_t \*const *instance* )

De-initializes the UART module.

This function de-initializes the UART module.

**Parameters**

<i>in</i>	<i>instance</i>	Pointer to the UART_PAL instance structure.
-----------	-----------------	---

**Returns**

STATUS\_SUCCESS: if successful; STATUS\_BUSY: if TX/RX line is busy; STATUS\_ERROR : if invalid instance type;

Definition at line 556 of file uart\_pal.c.

#### 16.135.4.4 status\_t UART\_GetBaudRate ( const uart\_instance\_t \*const *instance*, uint32\_t \* *configuredBaudRate* )

Returns the UART baud rate.

This function returns the UART configured baud rate.

**Parameters**

<i>in</i>	<i>instance</i>	Pointer to the UART_PAL instance structure.
<i>out</i>	<i>configuredBaudRate</i>	Pointer to configured baud rate.

**Returns**

STATUS\_SUCCESS: if successful; STATUS\_ERROR : if invalid instance type;

Definition at line 687 of file uart\_pal.c.

#### 16.135.4.5 void UART\_GetDefaultConfig ( uart\_user\_config\_t \* *config* )

Gets the default configuration structure.

This function gets the default configuration structure, with the following settings:

- Baud rate: 9600
- Number of bits in a character: 8
- Parity mode: disable
- Number of stop bits: 1 stop bit
- Type of the transfer: interrupt
- Callback to invoke for data receive: NULL
- Receive callback parameter: NULL
- Callback to invoke for data send: NULL
- Transmit callback parameter: NULL
- Setup pins for FLEXIO: NULL

#### Parameters

out	<i>config</i>	Pointer to the UART_PAL user configuration structure.
-----	---------------	---

#### Returns

NONE

Definition at line 396 of file `uart_pal.c`.

**16.135.4.6** `status_t UART_GetReceiveStatus ( const uart_instance_t *const instance, uint32_t * bytesRemaining )`

Get the status of the current non-blocking UART reception.

#### Parameters

in	<i>instance</i>	Pointer to the UART_PAL instance structure.
out	<i>bytesRemaining</i>	Pointer to value that is filled with the number of bytes that still need to be received in the active transfer

#### Note

In DMA mode, this parameter may not be accurate, in case the transfer completes right after calling this function; in this edge-case, the parameter will reflect the initial transfer size, due to automatic reloading of the major loop count in the DMA transfer descriptor.

return STATUS\_SUCCESS : The reception has completed successfully; STATUS\_BUSY : The reception is still in progress. *bytesReceived* will be filled with the number of bytes that have been received so far; STATUS\_UART\_RX\_OVERFLOW : If an overrun error occurred during the reception; STATUS\_UART\_ABORTED : The reception was aborted; STATUS\_TIMEOUT : A timeout was reached; STATUS\_ERROR : An error occurred; return For LPUART used by UART\_PAL: STATUS\_UART\_FRAMING\_ERROR: If bit stop on frame is wrong; STATUS\_UART\_PARITY\_ERROR : If bit parity on frame is wrong; STATUS\_UART\_NOISE\_ERROR : If noise happens on bus;

Definition at line 1062 of file `uart_pal.c`.

**16.135.4.7** `status_t UART_GetTransmitStatus ( const uart_instance_t *const instance, uint32_t * bytesRemaining )`

Get the status of the current non-blocking UART transmission.

**Parameters**

in	<i>instance</i>	Pointer to the UART_PAL instance structure.
out	<i>bytesRemaining</i>	Pointer to value that is populated with the number of bytes that have been sent in the active transfer.

**Note**

In DMA mode, this parameter may not be accurate, in case the transfer completes right after calling this function; in this edge-case, the parameter will reflect the initial transfer size, due to automatic reloading of the major loop count in the DMA transfer descriptor.

**Returns**

STATUS\_SUCCESS : The transmit has completed successfully; STATUS\_BUSY : The transmit is still in progress. *bytesTransmitted* will be filled with the number of bytes that have been transmitted so far; STATUS\_UART\_ABORTED : The transmit was aborted; STATUS\_TIMEOUT : A timeout was reached; STATUS\_ERROR : An error occurred;

Definition at line 876 of file `uart_pal.c`.

**16.135.4.8** `status_t UART_Init ( const uart_instance_t *const instance, const uart_user_config_t * config )`

Initializes the UART module.

This function initializes and enables the requested UART module, configuring the bus parameters.

**Parameters**

in	<i>instance</i>	Pointer to the UART_PAL instance structure.
in	<i>config</i>	Pointer to the UART_PAL user configuration structure.

return STATUS\_SUCCESS: if successful; STATUS\_ERROR : if invalid instance type or init over supported hardware. return For LPUART, LINFLEXD\_UART used by UART\_PAL: STATUS\_BUSY : if calling function while bus is busy;

Definition at line 419 of file `uart_pal.c`.

**16.135.4.9** `status_t UART_ReceiveData ( const uart_instance_t *const instance, uint8_t * rxBuff, uint32_t rxSize )`

Perform a non-blocking UART reception.

This function receives a block of data and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode).

**Parameters**

in	<i>instance</i>	Pointer to the UART_PAL instance structure.
out	<i>rxBuff</i>	Pointer to the data to be transferred.
in	<i>rxSize</i>	Length in bytes of the data to be transferred.

**Returns**

STATUS\_BUSY : if bus is busy; STATUS\_SUCCESS: if successful; STATUS\_ERROR : An error occurred;

Definition at line 972 of file `uart_pal.c`.

**16.135.4.10** `status_t UART_ReceiveDataBlocking ( const uart_instance_t *const instance, uint8_t * rxBuff, uint32_t rxSize, uint32_t timeout )`

Perform a blocking UART reception.

This function receives a block of data and only returns when the transmission is complete.

**Parameters**

in	<i>instance</i>	Pointer to the UART_PAL instance structure.
out	<i>rxBuff</i>	Pointer to the receive buffer.
in	<i>rxSize</i>	Length in bytes of the data to be received.
in	<i>timeout</i>	Timeout for the transfer in milliseconds.

return STATUS\_TIMEOUT : if waiting time for reception finished while receive data incompletely; STATUS\_BUSY : if bus is busy; STATUS\_SUCCESS : if successful; STATUS\_UART\_RX\_OVERRUN : If an overrun error occurred during the reception; STATUS\_ERROR : An error occurred; return For LPUART used by UART\_PAL: STATUS\_UART\_FRAMING\_ERROR: If bit stop on frame is wrong; STATUS\_UART\_PARITY\_ERROR : If bit parity on frame is wrong; STATUS\_UART\_NOISE\_ERROR : If noise happens on bus;

Definition at line 921 of file uart\_pal.c.

**16.135.4.11** `status_t UART_SendData ( const uart_instance_t *const instance, const uint8_t * txBuff, uint32_t txSize )`

Perform a non-blocking UART transmission.

This function sends a block of data and returns immediately. The rest of the transmission is handled by the interrupt service routine (if the driver is initialized in interrupt mode).

**Parameters**

in	<i>instance</i>	Pointer to the UART_PAL instance structure.
in	<i>txBuffer</i>	Pointer to the data to be transferred.
in	<i>txSize</i>	Length in bytes of the data to be transferred.

**Returns**

STATUS\_BUSY : if bus is busy; STATUS\_SUCCESS: if successful; STATUS\_ERROR : An error occurred;

Definition at line 785 of file uart\_pal.c.

**16.135.4.12** `status_t UART_SendDataBlocking ( const uart_instance_t *const instance, const uint8_t * txBuff, uint32_t txSize, uint32_t timeout )`

Perform a blocking UART transmission.

This function sends a block of data and only returns when the transmission is complete.

**Parameters**

in	<i>instance</i>	Pointer to the UART_PAL instance structure.
in	<i>txBuffer</i>	Pointer to the data to be transferred.
in	<i>txSize</i>	Length in bytes of the data to be transferred.
in	<i>timeout</i>	Timeout value in milliseconds.

**Returns**

STATUS\_TIMEOUT: if waiting time for transfer finished but transmit data incompletely; STATUS\_BUSY : if bus is busy; STATUS\_SUCCESS: if successful; STATUS\_ERROR : An error occurred;

Definition at line 732 of file uart\_pal.c.

**16.135.4.13** `status_t UART_SetBaudRate ( const uart_instance_t *const instance, uint32_t desiredBaudRate )`

Configures the UART baud rate.

This function configures the UART baud rate. Note that due to module limitation not any baud rate can be achieved. The driver will set a baud rate as close as possible to the requested baud rate, but there may still be substantial differences. The application should call [UART\\_GetBaudRate\(\)](#) after [UART\\_SetBaudRate\(\)](#) to check what baud rate was actually set.



**Parameters**

in	<i>instance</i>	Pointer to the UART_PAL instance structure.
in	<i>desiredBaudRate</i>	Desired baud rate.

**Returns**

STATUS\_SUCCESS: if successful; STATUS\_BUSY : if calling function while bus is busy; STATUS\_ERROR : if invalid instance type;

Definition at line 633 of file uart\_pal.c.

**16.135.4.14** `status_t UART_SetRxBuffer ( const uart_instance_t *const instance, uint8_t * rxBuff, uint32_t rxSize )`

Provide a buffer for receiving data.

The function can be used to provide a new buffer for receiving data to the driver. Beside, It can be called from rx callback to provide a new buffer for continuous reception.

**Parameters**

in	<i>instance</i>	Pointer to the UART_PAL instance structure.
in	<i>rxBuff</i>	Pointer to buffer containing received data.
in	<i>rxSize</i>	The number of bytes to receive.

**Returns**

STATUS\_SUCCESS: Provide completed; STATUS\_ERROR : if invalid instance type;

Definition at line 1107 of file uart\_pal.c.

**16.135.4.15** `status_t UART_SetTxBuffer ( const uart_instance_t *const instance, const uint8_t * txBuff, uint32_t txSize )`

Provide a buffer for transmitting data.

The function can be used to provide a new buffer for transmitting data to the driver. Beside, It can be called from tx callback to provide a new buffer for continuous transmission.

**Parameters**

in	<i>instance</i>	Pointer to the UART_PAL instance structure.
in	<i>txBuff</i>	Pointer to buffer containing transmitted data.
in	<i>txSize</i>	The number of bytes to transmit.

**Returns**

STATUS\_SUCCESS: Provide completed; STATUS\_ERROR : if invalid instance type;

Definition at line 1151 of file uart\_pal.c.

## 16.136 User provided call-outs

### 16.136.1 Detailed Description

This group contains APIs which may be called from within the LIN module in order to enable/disable LIN communication interrupts.

#### Functions

- `I_u16 I_sys_irq_disable (I_ifc_handle iii)`  
*Disable LIN related IRQ.*
- `void I_sys_irq_restore (I_ifc_handle iii)`  
*Enable LIN related IRQ.*

### 16.136.2 Function Documentation

#### 16.136.2.1 `I_u16 I_sys_irq_disable ( I_ifc_handle iii )`

Disable LIN related IRQ.

##### Parameters

<code>in</code>	<code>iii</code>	Interface name
-----------------	------------------	----------------

##### Returns

`I_u16`

Definition at line 543 of file `lin_common_api.c`.

#### 16.136.2.2 `void I_sys_irq_restore ( I_ifc_handle iii )`

Enable LIN related IRQ.

##### Parameters

<code>in</code>	<code>iii</code>	Interface name
-----------------	------------------	----------------

##### Returns

`void`

Definition at line 557 of file `lin_common_api.c`.

## 16.137 WDG PAL

### 16.137.1 Detailed Description

Watchdog Peripheral Abstraction Layer.

#### Data Structures

- struct [wdg\\_option\\_mode\\_t](#)  
WDG PAL option mode configuration structure Implements : [wdg\\_option\\_mode\\_t\\_Class](#). [More...](#)
- struct [extension\\_ewm\\_for\\_wdg\\_t](#)  
The extension structure for the EWM over WDOG peripheral Implements : [extension\\_ewm\\_for\\_wdg\\_t\\_Class](#). [More...](#)
- struct [wdg\\_config\\_t](#)  
WDG PAL configuration structure Implements : [wdg\\_config\\_t\\_Class](#). [More...](#)

#### Enumerations

- enum [wdg\\_clock\\_source\\_t](#) { [WDG\\_PAL\\_BUS\\_CLOCK](#) = 0x00U, [WDG\\_PAL\\_LPO\\_CLOCK](#) = 0x01U, [WDG\\_PAL\\_SOSC\\_CLOCK](#) = 0x02U, [WDG\\_PAL\\_SIRC\\_CLOCK](#) = 0x03U }
  - enum [wdg\\_in\\_assert\\_logic\\_t](#) { [WDG\\_IN\\_ASSERT\\_DISABLED](#) = 0x00U, [WDG\\_IN\\_ASSERT\\_ON\\_LOGIC\\_ZERO](#) = 0x01U, [WDG\\_IN\\_ASSERT\\_ON\\_LOGIC\\_ONE](#) = 0x02U }
  - enum [wdg\\_inst\\_type\\_t](#)
- Enumeration with the types of peripherals supported by Watchdog PAL.*

#### WDG PAL API

- status\_t [WDG\\_Init](#) (const [wdg\\_instance\\_t](#) \*const instance, const [wdg\\_config\\_t](#) \*configPtr)  
*Initializes the WDG PAL.*
- void [WDG\\_GetDefaultConfig](#) ([wdg\\_config\\_t](#) \*const config)  
*Gets default configuration of the WDG PAL.*
- void [WDG\\_Refresh](#) (const [wdg\\_instance\\_t](#) \*const instance)  
*Refreshes the WDG PAL counter.*
- status\_t [WDG\\_Deinit](#) (const [wdg\\_instance\\_t](#) \*const instance)  
*De-initializes the WDG PAL.*
- status\_t [WDG\\_SetInt](#) (const [wdg\\_instance\\_t](#) \*const instance, bool enable)  
*Set interrupt for WDG PAL.*
- status\_t [WDG\\_SetTimeout](#) (const [wdg\\_instance\\_t](#) \*const instance, uint32\_t value)  
*Sets the value of the WDG PAL timeout.*
- status\_t [WDG\\_SetWindow](#) (const [wdg\\_instance\\_t](#) \*const instance, bool enable, uint32\_t value)  
*Set window mode and window value of the WDG PAL.*
- status\_t [WDG\\_GetCounter](#) (const [wdg\\_instance\\_t](#) \*const instance, uint32\_t \*value)  
*Gets the value of the WDG PAL counter.*
- void [WDG\\_ClearIntFlag](#) (const [wdg\\_instance\\_t](#) \*const instance)  
*Clears the Timeout Interrupt Flag.*

## 16.137.2 Data Structure Documentation

### 16.137.2.1 struct wdg\_option\_mode\_t

WDG PAL option mode configuration structure Implements : wdg\_option\_mode\_t\_Class.

Definition at line 64 of file wdg\_pal.h.

#### Data Fields

- bool [wait](#)
- bool [stop](#)
- bool [debug](#)

#### Field Documentation

##### 16.137.2.1.1 bool debug

Debug mode

Definition at line 68 of file wdg\_pal.h.

##### 16.137.2.1.2 bool stop

Stop mode

Definition at line 67 of file wdg\_pal.h.

##### 16.137.2.1.3 bool wait

Wait mode

Definition at line 66 of file wdg\_pal.h.

### 16.137.2.2 struct extension\_ewm\_for\_wdg\_t

The extension structure for the EWM over WDOG peripheral Implements : extension\_ewm\_for\_wdg\_t\_Class.

Definition at line 88 of file wdg\_pal.h.

#### Data Fields

- [wdg\\_in\\_assert\\_logic\\_t](#) assertLogic
- [uint8\\_t](#) prescalerValue

#### Field Documentation

##### 16.137.2.2.1 wdg\_in\_assert\_logic\_t assertLogic

Assert logic for EWM input pin

Definition at line 90 of file wdg\_pal.h.

##### 16.137.2.2.2 uint8\_t prescalerValue

EWM clock prescaler

Definition at line 91 of file wdg\_pal.h.

### 16.137.2.3 struct wdg\_config\_t

WDG PAL configuration structure Implements : wdg\_config\_t\_Class.

Definition at line 99 of file wdg\_pal.h.

## Data Fields

- [wdg\\_clock\\_source\\_t](#) clkSource
- [wdg\\_option\\_mode\\_t](#) opMode
- [uint32\\_t](#) timeoutValue
- [uint8\\_t](#) percentWindow
- [bool](#) intEnable
- [bool](#) winEnable
- [bool](#) prescalerEnable
- [void \\*](#) extension

## Field Documentation

### 16.137.2.3.1 [wdg\\_clock\\_source\\_t](#) clkSource

The clock source of the WDOG

Definition at line 101 of file [wdg\\_pal.h](#).

### 16.137.2.3.2 [void\\*](#) extension

This field will be add extra settings to EWM's configuration

Definition at line 109 of file [wdg\\_pal.h](#).

### 16.137.2.3.3 [bool](#) intEnable

If true, an interrupt request is generated before reset

Definition at line 105 of file [wdg\\_pal.h](#).

### 16.137.2.3.4 [wdg\\_option\\_mode\\_t](#) opMode

The modes in which the WDOG is functional

Definition at line 102 of file [wdg\\_pal.h](#).

### 16.137.2.3.5 [uint8\\_t](#) percentWindow

The window value compares to timeout value. Maximum value is 100

Definition at line 104 of file [wdg\\_pal.h](#).

### 16.137.2.3.6 [bool](#) prescalerEnable

If true, prescaler is enabled( default prescaler = 256)

Definition at line 107 of file [wdg\\_pal.h](#).

### 16.137.2.3.7 [uint32\\_t](#) timeoutValue

The timeout value

Definition at line 103 of file [wdg\\_pal.h](#).

### 16.137.2.3.8 [bool](#) winEnable

If true, window mode is enabled

Definition at line 106 of file [wdg\\_pal.h](#).

## 16.137.3 Enumeration Type Documentation

### 16.137.3.1 enum wdg\_clock\_source\_t

Clock sources for the WDG PAL. Implements : wdg\_clock\_source\_t\_Class.

#### Enumerator

**WDG\_PAL\_BUS\_CLOCK** Bus clock  
**WDG\_PAL\_LPO\_CLOCK** LPO clock  
**WDG\_PAL\_SOSC\_CLOCK** SOSC clock  
**WDG\_PAL\_SIRC\_CLOCK** SIRC clock

Definition at line 52 of file wdg\_pal.h.

### 16.137.3.2 enum wdg\_in\_assert\_logic\_t

WDG PAL input pin configuration Configures if the input pin is enabled and when is asserted Implements : wdg\_in\_assert\_logic\_t\_Class.

#### Enumerator

**WDG\_IN\_ASSERT\_DISABLED** Input pin disabled  
**WDG\_IN\_ASSERT\_ON\_LOGIC\_ZERO** Input pin asserts EWM when on logic 0  
**WDG\_IN\_ASSERT\_ON\_LOGIC\_ONE** Input pin asserts EWM when on logic 1

Definition at line 77 of file wdg\_pal.h.

### 16.137.3.3 enum wdg\_inst\_type\_t

Enumeration with the types of peripherals supported by Watchdog PAL.

This enumeration contains the types of peripherals supported by Watchdog PAL. Implements : wdg\_inst\_type\_t\_Class

Definition at line 42 of file wdg\_pal\_mapping.h.

## 16.137.4 Function Documentation

### 16.137.4.1 void WDG\_ClearIntFlag ( const wdg\_instance\_t \*const instance )

Clears the Timeout Interrupt Flag.

This function clears the Timeout Interrupt Flag.

#### Parameters

in	instance	The name of the instance.
----	----------	---------------------------

Definition at line 488 of file wdg\_pal.c.

### 16.137.4.2 status\_t WDG\_Deinit ( const wdg\_instance\_t \*const instance )

De-initializes the WDG PAL.

This function resets all configuration to default and disable the WDG PAL instance.

#### Parameters

<i>in</i>	<i>instance</i>	The name of the instance.
-----------	-----------------	---------------------------

**Returns**

operation status

- STATUS\_SUCCESS : Operation was successful.
- STATUS\_ERROR : Operation failed due to WDG PAL was locked.
- STATUS\_UNSUPPORTED : Operation was unsupported.

Definition at line 282 of file wdg\_pal.c.

**16.137.4.3 status\_t WDG\_GetCounter ( const wdg\_instance\_t \*const instance, uint32\_t \* value )**

Gets the value of the WDG PAL counter.

This function gets counter of WDG PAL module. Note that: Counter will be reset to timeout value if WDG PAL uses SWT. The counter will continue to run if WDG PAL uses WDOG.

**Parameters**

<i>in</i>	<i>instance</i>	The name of the instance.
<i>out</i>	<i>value</i>	Pointer to the counter value

**Returns**

operation status

- STATUS\_SUCCESS : Operation was successful.
- STATUS\_ERROR : Operation failed due to SWT was lock by hard lock.
- STATUS\_UNSUPPORTED : Operation was unsupported.

Definition at line 440 of file wdg\_pal.c.

**16.137.4.4 void WDG\_GetDefaultConfig ( wdg\_config\_t \*const config )**

Gets default configuration of the WDG PAL.

This function gets the default configuration of the WDG PAL.

**Parameters**

<i>out</i>	<i>configures</i>	the default configuration
------------	-------------------	---------------------------

Definition at line 206 of file wdg\_pal.c.

**16.137.4.5 status\_t WDG\_Init ( const wdg\_instance\_t \*const instance, const wdg\_config\_t \* configPtr )**

Initializes the WDG PAL.

This function initializes the WDG instance by user configuration

**Parameters**

<i>in</i>	<i>instance</i>	The name of the instance.
<i>in</i>	<i>configPtr</i>	Pointer to the WDG PAL user configuration structure

**Returns**

operation status

- STATUS\_SUCCESS : Operation was successful.
- STATUS\_ERROR : Operation failed. Possible causes: previous clock source or the one specified in the configuration structure is disabled; WDG PAL configuration updates are not allowed.

- STATUS\_UNSUPPORTED : Operation was unsupported.

Definition at line 80 of file wdg\_pal.c.

#### 16.137.4.6 void WDG\_Refresh ( const wdg\_instance\_t \*const instance )

Refreshes the WDG PAL counter.

This function resets the WDG PAL counter

##### Parameters

in	instance	The name of the instance.
----	----------	---------------------------

Definition at line 243 of file wdg\_pal.c.

#### 16.137.4.7 status\_t WDG\_SetInt ( const wdg\_instance\_t \*const instance, bool enable )

Set interrupt for WDG PAL.

This function enables/disables the WDG PAL timeout interrupt and sets a function to be called when a timeout interrupt is received, before reset.

##### Parameters

in	instance	The name of the instance.
in	enable	<ul style="list-style-type: none"> <li>• true : Enable interrupt</li> <li>• false : Disable interrupt</li> </ul>

##### Returns

operation status

- STATUS\_SUCCESS : Operation was successful.
- STATUS\_ERROR : Operation failed. Possible causes: failed to WDG PAL configuration updates not allowed.
- STATUS\_UNSUPPORTED: Operation was unsupported.

Definition at line 319 of file wdg\_pal.c.

#### 16.137.4.8 status\_t WDG\_SetTimeout ( const wdg\_instance\_t \*const instance, uint32\_t value )

Sets the value of the WDG PAL timeout.

This function sets the value of the WDG PAL timeout.

##### Parameters

in	instance	The name of the instance.
in	value	The value of the WDG PAL timeout.

##### Returns

operation status

- STATUS\_SUCCESS : Operation was successful.
- STATUS\_ERROR : Operation failed due to WDG PAL was locked.
- STATUS\_UNSUPPORTED : Operation was unsupported.

Definition at line 358 of file wdg\_pal.c.



16.137.4.9 `status_t WDG_SetWindow ( const wdg_instance_t *const instance, bool enable, uint32_t value )`

Set window mode and window value of the WDG PAL.

This function set window mode, window value is set when window mode enabled.

#### Parameters

in	<i>instance</i>	The name of the instance.
in	<i>enable</i>	<ul style="list-style-type: none"><li>• true : Enable window mode</li><li>• false : Disable window mode</li></ul>
in	<i>value</i>	The value of the WDG PAL window.

#### Returns

operation status

- STATUS\_SUCCESS : Operation was successful.
- STATUS\_ERROR : Operation failed due to WDG PAL was locked.
- STATUS\_UNSUPPORTED : Operation was unsupported.

Definition at line 397 of file wdg\_pal.c.

## 16.138 WDOG Driver

### 16.138.1 Detailed Description

Watchdog Timer Peripheral Driver.

How to use the WDOG driver in your application

In order to be able to use the Watchdog in your application, the first thing to do is initializing it with the desired configuration. This is done by calling the **WDOG\_DRV\_Init** function. One of the arguments passed to this function is the configuration which will be used for the Watchdog, specified by the **wdog\_user\_config\_t** structure.

The **wdog\_user\_config\_t** structure allows you to configure the following:

- the clock source of the Watchdog;
- the prescaler (a fixed 256 pre-scaling of the Watchdog counter reference clock may be enabled);
- the operation modes in which the Watchdog is functional (by default, the Watchdog is not functional in Debug mode, Wait mode or Stop mode);
- the timeout value to which the Watchdog counter is compared;
- the window mode option for the refresh mechanism (by default, the window mode is disabled, but it may be enabled and a window value may be set);
- the Watchdog timeout interrupt (if enabled, after a reset-triggering event, the Watchdog first generates an interrupt request; next, the Watchdog delays 128 bus clocks before forcing a reset, to allow the interrupt service routine to perform tasks (like analyzing the stack to debug code));
- the update mechanism (by default, the Watchdog reconfiguration is enabled, but updates can be disabled)

**Please note** that if the updates are disabled the Watchdog cannot be later modified without forcing a reset (this implies that further calls of the **WDOG\_DRV\_Init**, **WDOG\_DRV\_Deinit** or **WDOG\_DRV\_SetInt** functions will lead to a reset).

As mentioned before, a timeout interrupt may be enabled by specifying it at the module initialization. The **WDOG\_DRV\_Init** only allows enabling/disabling the interrupt, and it does not set up the ISR to be used for the interrupt request. In order to set up a function to be called after a reset-triggering event (and also enable/disable the interrupt), the **WDOG\_DRV\_SetInt** function may be used. **Please note** that, due to the 128 bus clocks delay before the reset, a limited amount of job can be done in the ISR.

#### Integration guideline

#### Compilation units

The following files need to be compiled in the project:

```
{S32SDK_PATH}\platform\drivers\src\wdog\wdog_driver.c
{S32SDK_PATH}\platform\drivers\src\wdog\wdog_hw_driver.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
{S32SDK_PATH}\platform\drivers\inc\
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

[Clock Manager Interrupt Manager \(Interrupt\)](#)

## Basic Operations of WDOG

1. To initialize WDOG, call `WDOG_DRV_Init()` with an user configuration structure. In the following code, WDOG is initialized with default settings.

```
#define INST_WDOG1 (0U)

wdog_user_config_t userConfigPtr = {
    .WD OG_LPO_CLOCK,      /* Use the LPO clock as source */
    .opMode = {            /* WDOG not functional in Wait/Debug/Stop mode */
        false,
        false,
        false
    },
    .true,                 /* Enable further updates of the WDOG configuration */
    .false,                 /* Timeout interrupt disabled */
    .false,                 /* Window mode disabled */
    .0U,                   /* Window value */
    .0x400,                 /* Timeout value */
    .false                  /* Prescaler disabled */
};

/* Initialize WDOG module */
WDOG_DRV_Init(INST_WDOG1, &userConfigPtr);
```

2. To get default configuration of WDOG module, just call the function `WDOG_DRV_GetDefaultConfig()`. Make sure that the operation before WDOG timeout executing.

```
wdog_user_config_t userConfigPtr;

/* Get default configuration of WDOG module */
WDOG_DRV_GetDefaultConfig(&userConfigPtr);
```

3. To refresh WDOG counter of WDOG module, just call the function `WDOG_DRV_Trigger()`. Make sure that the operation before WDOG timeout executing.

```
/* Refresh counter of WDOG counter */
WDOG_DRV_Trigger(INST_WDOG1);
```

4. To de-initialize WDOG module, just call the function `WDOG_DRV_Deinit()`. Make sure that the operation before WDOG timeout executing.

```
/* De-initialize WDOG module */
WDOG_DRV_Deinit(INST_WDOG1);
```

## Example:

```
#define INST_WDOG1 (0U)

wdog_user_config_t userConfigPtr = {
    .WD OG_LPO_CLOCK,      /* Use the LPO clock as source */
    .opMode = {            /* WDOG not functional in Wait/Debug/Stop mode */
        false,
        false,
        false
    },
    .true,                 /* Enable further updates of the WDOG configuration */
    .false,                 /* Timeout interrupt disabled */
    .false,                 /* Window mode disabled */
    .0U,                   /* Window value */
    .0x400,                 /* Timeout value */
    .false                  /* Prescaler disabled */
};

/* Initialize WDOG module */
WDOG_DRV_Init(INST_WDOG1, &userConfigPtr);

/* Enable the timeout interrupt and set the ISR */
WDOG_DRV_SetInt(INST_WDOG1, true);

while (1) {

    /* Do something that takes between 0x100 and 0x400 clock cycles */

    /* Refresh the counter */
    WDOG_DRV_Trigger(INST_WDOG1);
}

/* De-initialize WDOG module */
WDOG_DRV_Deinit(INST_WDOG1);
```

## Data Structures

- struct [wdog\\_op\\_mode\\_t](#)  
WDOG option mode configuration structure Implements : [wdog\\_op\\_mode\\_t\\_Class](#). [More...](#)
- struct [wdog\\_user\\_config\\_t](#)  
WDOG user configuration structure Implements : [wdog\\_user\\_config\\_t\\_Class](#). [More...](#)

## Enumerations

- enum [wdog\\_clk\\_source\\_t](#) { [WDOG\\_BUS\\_CLOCK](#) = 0x00U, [WDOG\\_LPO\\_CLOCK](#) = 0x01U, [WDOG\\_SOSC\\_CLOCK](#) = 0x02U, [WDOG\\_SIRC\\_CLOCK](#) = 0x03U }  
Clock sources for the WDOG. Implements : [wdog\\_clk\\_source\\_t\\_Class](#).
- enum [wdog\\_test\\_mode\\_t](#) { [WDOG\\_TST\\_DISABLED](#) = 0x00U, [WDOG\\_TST\\_USER](#) = 0x01U, [WDOG\\_TST\\_LOW](#) = 0x02U, [WDOG\\_TST\\_HIGH](#) = 0x03U }  
Test modes for the WDOG. Implements : [wdog\\_test\\_mode\\_t\\_Class](#).
- enum [wdog\\_set\\_mode\\_t](#) { [WDOG\\_DEBUG\\_MODE](#) = 0x00U, [WDOG\\_WAIT\\_MODE](#) = 0x01U, [WDOG\\_STOP\\_MODE](#) = 0x02U }  
set modes for the WDOG. Implements : [wdog\\_set\\_mode\\_t\\_Class](#)

## WDOG Driver API

- status\_t [WDOG\\_DRV\\_Init](#) (uint32\_t instance, const [wdog\\_user\\_config\\_t](#) \*userConfigPtr)  
Initializes the WDOG driver.
- status\_t [WDOG\\_DRV\\_Deinit](#) (uint32\_t instance)  
De-initializes the WDOG driver.
- void [WDOG\\_DRV\\_GetConfig](#) (uint32\_t instance, [wdog\\_user\\_config\\_t](#) \*const config)  
Gets the current configuration of the WDOG.
- void [WDOG\\_DRV\\_GetDefaultConfig](#) ([wdog\\_user\\_config\\_t](#) \*const config)  
Gets default configuration of the WDOG.
- status\_t [WDOG\\_DRV\\_SetInt](#) (uint32\_t instance, bool enable)  
Enables/Disables the WDOG timeout interrupt and sets a function to be called when a timeout interrupt is received, before reset.
- void [WDOG\\_DRV\\_ClearIntFlag](#) (uint32\_t instance)  
Clear interrupt flag of the WDOG.
- void [WDOG\\_DRV\\_Trigger](#) (uint32\_t instance)  
Refreshes the WDOG counter.
- uint16\_t [WDOG\\_DRV\\_GetCounter](#) (uint32\_t instance)  
Gets the value of the WDOG counter.
- status\_t [WDOG\\_DRV\\_SetWindow](#) (uint32\_t instance, bool enable, uint16\_t windowvalue)  
Set window mode and window value of the WDOG.
- status\_t [WDOG\\_DRV\\_SetMode](#) (uint32\_t instance, bool enable, [wdog\\_set\\_mode\\_t](#) Setmode)  
Sets the mode operation of the WDOG.
- status\_t [WDOG\\_DRV\\_SetTimeout](#) (uint32\_t instance, uint16\_t timeout)  
Sets the value of the WDOG timeout.
- status\_t [WDOG\\_DRV\\_SetTestMode](#) (uint32\_t instance, [wdog\\_test\\_mode\\_t](#) testMode)  
Changes the WDOG test mode.
- [wdog\\_test\\_mode\\_t](#) [WDOG\\_DRV\\_GetTestMode](#) (uint32\_t instance)  
Gets the WDOG test mode.

## 16.138.2 Data Structure Documentation

### 16.138.2.1 struct wdog\_op\_mode\_t

WDOG option mode configuration structure Implements : wdog\_op\_mode\_t\_Class.

Definition at line 84 of file wdog\_driver.h.

#### Data Fields

- bool [wait](#)
- bool [stop](#)
- bool [debug](#)

#### Field Documentation

##### 16.138.2.1.1 bool debug

Debug mode

Definition at line 88 of file wdog\_driver.h.

##### 16.138.2.1.2 bool stop

Stop mode

Definition at line 87 of file wdog\_driver.h.

##### 16.138.2.1.3 bool wait

Wait mode

Definition at line 86 of file wdog\_driver.h.

### 16.138.2.2 struct wdog\_user\_config\_t

WDOG user configuration structure Implements : wdog\_user\_config\_t\_Class.

Definition at line 95 of file wdog\_driver.h.

#### Data Fields

- [wdog\\_clk\\_source\\_t](#) clkSource
- [wdog\\_op\\_mode\\_t](#) opMode
- bool [updateEnable](#)
- bool [intEnable](#)
- bool [winEnable](#)
- [uint16\\_t](#) [windowValue](#)
- [uint16\\_t](#) [timeoutValue](#)
- bool [prescalerEnable](#)

#### Field Documentation

##### 16.138.2.2.1 wdog\_clk\_source\_t clkSource

The clock source of the WDOG

Definition at line 97 of file wdog\_driver.h.

##### 16.138.2.2.2 bool intEnable

If true, an interrupt request is generated before reset

Definition at line 100 of file wdog\_driver.h.

**16.138.2.2.3 wdog\_op\_mode\_t opMode**

The modes in which the WDOG is functional

Definition at line 98 of file wdog\_driver.h.

**16.138.2.2.4 bool prescalerEnable**

If true, a fixed 256 prescaling of the counter reference clock is enabled

Definition at line 104 of file wdog\_driver.h.

**16.138.2.2.5 uint16\_t timeoutValue**

The timeout value

Definition at line 103 of file wdog\_driver.h.

**16.138.2.2.6 bool updateEnable**

If true, further updates of the WDOG are enabled

Definition at line 99 of file wdog\_driver.h.

**16.138.2.2.7 uint16\_t windowValue**

The window value

Definition at line 102 of file wdog\_driver.h.

**16.138.2.2.8 bool winEnable**

If true, window mode is enabled

Definition at line 101 of file wdog\_driver.h.

**16.138.3 Enumeration Type Documentation****16.138.3.1 enum wdog\_clk\_source\_t**

Clock sources for the WDOG. Implements : wdog\_clk\_source\_t\_Class.

Enumerator

**WDOG\_BUS\_CLOCK** Bus clock

**WDOG\_LPO\_CLOCK** LPO clock

**WDOG\_SOSC\_CLOCK** SOSC clock

**WDOG\_SIRC\_CLOCK** SIRC clock

Definition at line 49 of file wdog\_driver.h.

**16.138.3.2 enum wdog\_set\_mode\_t**

set modes for the WDOG. Implements : wdog\_set\_mode\_t\_Class

Enumerator

**WDOG\_DEBUG\_MODE** Debug mode

**WDOG\_WAIT\_MODE** Wait mode

**WDOG\_STOP\_MODE** Stop mode

Definition at line 73 of file wdog\_driver.h.

## 16.138.3.3 enum wdog\_test\_mode\_t

Test modes for the WDOG. Implements : wdog\_test\_mode\_t\_Class.

## Enumerator

**WDOG\_TST\_DISABLED** Test mode disabled

**WDOG\_TST\_USER** User mode enabled. (Test mode disabled.)

**WDOG\_TST\_LOW** Test mode enabled, only the low byte is used.

**WDOG\_TST\_HIGH** Test mode enabled, only the high byte is used.

Definition at line 61 of file wdog\_driver.h.

## 16.138.4 Function Documentation

## 16.138.4.1 void WDOG\_DRV\_ClearIntFlag ( uint32\_t instance )

Clear interrupt flag of the WDOG.

## Parameters

in	instance	WDOG peripheral instance number
----	----------	---------------------------------

Definition at line 270 of file wdog\_driver.c.

## 16.138.4.2 status\_t WDOG\_DRV\_Deinit ( uint32\_t instance )

De-initializes the WDOG driver.

## Parameters

in	instance	WDOG peripheral instance number
----	----------	---------------------------------

## Returns

operation status

- STATUS\_SUCCESS: if allowed reconfigures WDOG module and de-initializes successful.
- STATUS\_ERROR: Operation failed. Possible causes: failed to WDOG configuration updates not allowed.

Definition at line 160 of file wdog\_driver.c.

## 16.138.4.3 void WDOG\_DRV\_GetConfig ( uint32\_t instance, wdog\_user\_config\_t \*const config )

Gets the current configuration of the WDOG.

## Parameters

in	instance	WDOG peripheral instance number
out	configures	the current configuration

Definition at line 195 of file wdog\_driver.c.

## 16.138.4.4 uint16\_t WDOG\_DRV\_GetCounter ( uint32\_t instance )

Gets the value of the WDOG counter.

**Parameters**

<i>in</i>	<i>instance</i>	WDOG peripheral instance number.
-----------	-----------------	----------------------------------

**Returns**

the value of the WDOG counter.

Definition at line 301 of file wdog\_driver.c.

**16.138.4.5** void WDOG\_DRV\_GetDefaultConfig ( wdog\_user\_config\_t \*const config )

Gets default configuration of the WDOG.

**Parameters**

<i>out</i>	<i>configures</i>	the default configuration
------------	-------------------	---------------------------

Definition at line 212 of file wdog\_driver.c.

**16.138.4.6** wdog\_test\_mode\_t WDOG\_DRV\_GetTestMode ( uint32\_t instance )

Gets the WDOG test mode.

This function verifies the test mode of the WDOG.

**Parameters**

<i>in</i>	<i>instance</i>	WDOG peripheral instance number
-----------	-----------------	---------------------------------

**Returns**

Test modes for the WDOG

Definition at line 459 of file wdog\_driver.c.

**16.138.4.7** status\_t WDOG\_DRV\_Init ( uint32\_t instance, const wdog\_user\_config\_t \* userConfigPtr )

Initializes the WDOG driver.

**Parameters**

<i>in</i>	<i>instance</i>	WDOG peripheral instance number
<i>in</i>	<i>userConfigPtr</i>	pointer to the WDOG user configuration structure

**Returns**

operation status

- STATUS\_SUCCESS: Operation was successful.
- STATUS\_ERROR: Operation failed. Possible causes: previous clock source or the one specified in the configuration structure is disabled; WDOG configuration updates are not allowed; WDOG instance has been initialized before; If window mode enabled and window value greater than or equal to the timeout value.

Definition at line 102 of file wdog\_driver.c.

**16.138.4.8** status\_t WDOG\_DRV\_SetInt ( uint32\_t instance, bool enable )

Enables/Disables the WDOG timeout interrupt and sets a function to be called when a timeout interrupt is received, before reset.



**Parameters**

in	<i>instance</i>	WDOG peripheral instance number
in	<i>enable</i>	enable/disable interrupt

**Returns**

operation status

- STATUS\_SUCCESS: if allowed reconfigures WDOG timeout interrupt.
- STATUS\_ERROR: Operation failed. Possible causes: failed to WDOG configuration updates not allowed.

Definition at line 238 of file wdog\_driver.c.

**16.138.4.9** `status_t WDOG_DRV_SetMode ( uint32_t instance, bool enable, wdog_set_mode_t Setmode )`

Sets the mode operation of the WDOG.

This function changes the mode operation of the WDOG.

**Parameters**

in	<i>instance</i>	WDOG peripheral instance number.
in	<i>enable</i>	enable/disable mode of the WDOG.
in	<i>Setmode</i>	select mode of the WDOG.

**Returns**

operation status

- STATUS\_SUCCESS: if allowed reconfigures mode operation of the WDOG.
- STATUS\_ERROR: Operation failed. Possible causes: failed to WDOG configuration updates not allowed.

Definition at line 352 of file wdog\_driver.c.

**16.138.4.10** `status_t WDOG_DRV_SetTestMode ( uint32_t instance, wdog_test_mode_t testMode )`

Changes the WDOG test mode.

This function changes the test mode of the WDOG. If the WDOG is tested in mode, software should set this field to 0x01U in order to indicate that the WDOG is functioning normally.

**Parameters**

in	<i>instance</i>	WDOG peripheral instance number
in	<i>testMode</i>	Test modes for the WDOG.

**Returns**

operation status

- STATUS\_SUCCESS: if allowed reconfigures WDOG test mode.
- STATUS\_ERROR: Operation failed. Possible causes: failed to WDOG configuration updates not allowed.

Definition at line 426 of file wdog\_driver.c.

**16.138.4.11** `status_t WDOG_DRV_SetTimeout ( uint32_t instance, uint16_t timeout )`

Sets the value of the WDOG timeout.

This function sets the value of the WDOG timeout.

**Parameters**

in	<i>instance</i>	WDOG peripheral instance number.
in	<i>timeout</i>	the value of the WDOG timeout.

**Returns**

operation status

- STATUS\_SUCCESS: if allowed reconfigures WDOG timeout.
- STATUS\_ERROR: Operation failed. Possible causes: failed to WDOG configuration updates not allowed.

Definition at line 397 of file wdog\_driver.c.

**16.138.4.12** `status_t WDOG_DRV_SetWindow ( uint32_t instance, bool enable, uint16_t windowvalue )`

Set window mode and window value of the WDOG.

This function set window mode, window value is set when window mode enabled.

**Parameters**

in	<i>instance</i>	WDOG peripheral instance number.
in	<i>enable</i>	enable/disable window mode and window value.
in	<i>windowvalue</i>	the value of the WDOG window.

**Returns**

operation status

- STATUS\_SUCCESS: if allowed reconfigures window value success.
- STATUS\_ERROR: Operation failed. Possible causes: failed to WDOG configuration updates not allowed.

Definition at line 316 of file wdog\_driver.c.

**16.138.4.13** `void WDOG_DRV_Trigger ( uint32_t instance )`

Refreshes the WDOG counter.

**Parameters**

in	<i>instance</i>	WDOG peripheral instance number
----	-----------------	---------------------------------

Definition at line 286 of file wdog\_driver.c.

## 16.139 Watchdog Peripheral Abstraction Layer (WDG PAL)

### 16.139.1 Detailed Description

The S32 SDK provides a Peripheral Abstraction Layer for Watchdog (WDG PAL) modules of S32 SDK devices.

The Watchdog PAL driver allows to generate interrupt event to reset CPU or external circuit. It was designed to be portable across all platforms and IPs which support Watchdog Timer.

#### Integration guideline

##### Define IPs specification

Unlike the other drivers, WDG PAL modules need to include a configuration file named `wdg_pal_cfg.h`, which allows the user to specify which IPs are used and how many resources are allocated for each of them (state structures). The following code example shows how to configure one instance for each available WDG IPs.

```
#ifndef WDG_PAL_CFG_H
#define WDG_PAL_CFG_H

/* Define which IP instance will be used in current project */
#define WDG_OVER_WDOG
#define WDG_OVER_EWM
#define WDG_OVER_SWT

/* Define the resources necessary for current project */
#define WDG_OVER_WDOG_INSTANCE_COUNT 1U
#define WDG_OVER_EWM_INSTANCE_COUNT 1U
#define WDG_OVER_SWT_INSTANCE_COUNT 0U

#endif /* WDG_PAL_CFG_H */
```

The following table contains the matching between platforms and available IPs

IP/MCU	S32K11x	S32K14x	MPC574x	S32Rx7x
WDOG	YES	YES	NO	NO
EWM	NO	YES	NO	NO
SWT	NO	NO	YES	YES

#### Compilation units

The following files need to be compiled in the project:

```
${S32SDK_PATH}\platform\pal\src\wdg\wdg_pal.c
```

#### Include path

The following paths need to be added to the include path of the toolchain:

```
${S32SDK_PATH}\platform\pal\inc
```

#### Compile symbols

No special symbols are required for this component

#### Dependencies

- [Clock Manager](#)
- [Interrupt Manager \(Interrupt\)](#)
- [External Watchdog Monitor \(EWM\)](#)

- [Watchdog timer \(WDOG\)](#)
- [swt](#)

## Functionality

### Initialization

In order to use the WDG PAL driver it must be first initialized, using [WDG\\_Init\(\)](#) function. Once initialized, it cannot be initialized again for the same WDG module instance until it is de-initialized, using [WDG\\_Deinit\(\)](#). Different WDG modules instances can function independently of each other.

### Interrupt event

After initialization, WDG PAL counter will count to timeout value. In window mode, when WDG PAL counter is refreshed, it will reset count to default value and count again. If WDG PAL counter count to timeout value, CPU or the external circuit will be reseted or placed into safe mode.

The configuration structure includes a special field named extension. It will be used only for WDG PAL over EWM peripheral and should contain a pointer to [extension\\_ewm\\_for\\_wdg\\_t](#) structure. The purpose of this structure is to configure which EWM\_OUT pins and clock prescaler are used by the applications.

### WDG PAL internal counter

WDG PAL internal counter is

- 8 bit if WDG PAL uses EWM
- 16 bit if WDG PAL uses WDOG
- 32 bit if WDG PAL uses SWT

WDG PAL's counter over EWM and WDOG will start to count from 0 to timeout value. WDG PAL's counter over SWT will start to count from timeout value to 0.

### Important Notes

- Before using the WDG PAL driver the module clock must be configured. Refer to Clock Manager for clock configuration.
- The driver enables the interrupts for the corresponding WDG PAL module, but any interrupt priority must be done by the application
- For WDG PAL over SWT, if the counter clock is slow, the software needs a wait time (inversely proportional to counter clock frequency) to synchronization completed.

### Example code

```
const wdg_instance_t wdg_pal1_Instance =
{
    .instType = WDG_INST_TYPE_WDOG,
    .instIdx  = 0U
};

const wdg_instance_t wdg_pal2_Instance =
{
    .instType = WDG_INST_TYPE_EWM,
    .instIdx  = 0U
};

const wdg_instance_t wdg_pal3_Instance =
{
    .instType = WDG_INST_TYPE_SWT,
    .instIdx  = 0U
};
```

```

/* Serial User Configurations */

const wdg_config_t wdg_pal1_Config0 =
{
    .clkSource      = WDG_PAL_LPO_CLOCK,
    .opMode         =
    {
        .wait       = false,
        .stop       = false,
        .debug      = false
    },
    .timeoutValue   = 1024,
    .percentWindow  = 50,
    .intEnable      = true,
    .winEnable      = true,
    .prescalerEnable = true
};

const wdg_config_t wdg_pal2_Config0 =
{
    .clkSource      = WDG_PAL_LPO_CLOCK,
    .opMode         =
    {
        .wait       = false,
        .stop       = false,
        .debug      = false
    },
    .timeoutValue   = 254,
    .percentWindow  = 100,
    .intEnable      = true,
    .winEnable      = true,
    .prescalerEnable = true,
    .extension      = &wdg_pal2_Extension0
};

extension_ewm_for_wdg_t wdg_pal2_Extension0 =
{
    .assertLogic    = WDG_IN_ASSERT_ON_LOGIC_ZERO,
    .prescalerValue = 251
};

const wdg_config_t wdg_pal3_Config0 =
{
    .clkSource      = WDG_PAL_LPO_CLOCK,
    .opMode         =
    {
        .wait       = false,
        .stop       = false,
        .debug      = false
    },
    .timeoutValue   = 2560,
    .percentWindow  = 50,
    .intEnable      = true,
    .winEnable      = true,
    .prescalerEnable = false
};

int main()
{
    /* Init clocks, pins, led and other modules */
    ...

    /* Initialize WDG PAL */
    WDG_Init(&wdg_pal1_Instance, &wdg_pal1_Config0);

    /* Infinite loop*/
    while(1)
    {
        /* Do something until the counter needs to be refreshed */
        ...
        /* Reset WDG PAL counter */
        WDG_Refresh(&wdg_pal1_Instance);
    }
}

```

## Modules

- [WDG PAL](#)

*Watchdog Peripheral Abstraction Layer.*

## 16.140 Watchdog timer (WDOG)

### 16.140.1 Detailed Description

The S32 SDK provides Peripheral Driver for the Watchdog timer (WDOG) module of S32 SDK devices.

#### Hardware background

The Watchdog Timer (WDOG) module is an independent timer that is available for system use. It provides a safety feature to ensure that software is executing as planned and that the CPU is not stuck in an infinite loop or executing unintended code. If the WDOG module is not serviced (refreshed) within a certain period, it resets the MCU.

Features of the WDOG module include:

- Configurable clock source inputs independent from the bus clock
- Programmable timeout period
  - Programmable 16-bit timeout value
  - Optional fixed 256 clock prescaler when longer timeout periods are needed
- Window mode option for the refresh mechanism
  - Programmable 16-bit window value
  - Provides robust check that program flow is faster than expected
  - Early refresh attempts trigger a reset
- Optional timeout interrupt to allow post-processing diagnostics
  - Interrupt request to CPU with interrupt vector for an interrupt service routine (ISR)
  - Forced reset occurs 128 bus clocks after the interrupt vector fetch
- Configuration bits are write-once-after-reset to ensure watchdog configuration cannot be mistakenly altered
- Robust write sequence for unlocking write-once configuration bits

#### Modules

- [WDOG Driver](#)  
*Watchdog Timer Peripheral Driver.*

## 17 Data Structure Documentation

### 17.1 `adc_callback_info_t` Struct Reference

Defines a structure used to pass information to the ADC PAL callback.

```
#include <platform/devices/callbacks.h>
```

#### Data Fields

- `uint32_t` [groupIndex](#)
- `uint16_t` [resultBufferTail](#)

#### 17.1.1 Detailed Description

Defines a structure used to pass information to the ADC PAL callback.

Implements : `adc_callback_info_t_Class`

Definition at line 108 of file `callbacks.h`.

#### 17.1.2 Field Documentation

##### 17.1.2.1 `uint32_t` [groupIndex](#)

Index of the group executing the callback.

Definition at line 110 of file `callbacks.h`.

##### 17.1.2.2 `uint16_t` [resultBufferTail](#)

Offset of the most recent conversion result in the result buffer.

Definition at line 111 of file `callbacks.h`.

The documentation for this struct was generated from the following file:

- `platform/devices/callbacks.h`

### 17.2 `adc_instance_t` Struct Reference

Structure storing PAL instance information.

```
#include <platform/pal/inc/adc_pal_mapping.h>
```

#### Data Fields

- `adc_inst_type_t` [instType](#)
- `uint32_t` [instIdx](#)

#### 17.2.1 Detailed Description

Structure storing PAL instance information.

This structure is used for storing PAL instance information. Implements : `adc_instance_t_Class`

Definition at line 74 of file `adc_pal_mapping.h`.

### 17.2.2 Field Documentation

#### 17.2.2.1 uint32\_t instIdx

Instance index of the peripheral (for ADC PAL the triggering peripheral) over which the PAL is used

Definition at line 77 of file `adc_pal_mapping.h`.

#### 17.2.2.2 adc\_inst\_type\_t instType

Peripheral over which the PAL is used

Definition at line 76 of file `adc_pal_mapping.h`.

The documentation for this struct was generated from the following file:

- `platform/pal/inc/adc_pal_mapping.h`

### 17.3 can\_instance\_t Struct Reference

Structure storing PAL instance information.

```
#include <platform/pal/inc/can_pal_mapping.h>
```

#### Data Fields

- `can_inst_type_t instType`
- `uint32_t instIdx`

#### 17.3.1 Detailed Description

Structure storing PAL instance information.

This structure is used for storing PAL instance information. Implements : `can_instance_t_Class`

Definition at line 54 of file `can_pal_mapping.h`.

### 17.3.2 Field Documentation

#### 17.3.2.1 uint32\_t instIdx

Instance index of the peripheral over which the PAL is used

Definition at line 56 of file `can_pal_mapping.h`.

#### 17.3.2.2 can\_inst\_type\_t instType

Peripheral over which the PAL is used

Definition at line 55 of file `can_pal_mapping.h`.

The documentation for this struct was generated from the following file:

- `platform/pal/inc/can_pal_mapping.h`

### 17.4 drv\_config\_t Struct Reference

#### Data Fields

- `sbc_wtdog_ctr_t watchdogCtr`



- uint32\_t [lpspiIntace](#)
- bool [isInit](#)

#### 17.4.1 Detailed Description

Definition at line 58 of file `sbc_uja116x_driver.c`.

#### 17.4.2 Field Documentation

##### 17.4.2.1 bool isInit

Definition at line 61 of file `sbc_uja116x_driver.c`.

##### 17.4.2.2 uint32\_t lpspiIntace

Definition at line 60 of file `sbc_uja116x_driver.c`.

##### 17.4.2.3 sbc\_wtdog\_ctr\_t watchdogCtr

Definition at line 59 of file `sbc_uja116x_driver.c`.

The documentation for this struct was generated from the following file:

- `middleware/sbc/sbc_uja116x/source/sbc_uja116x_driver.c`

## 17.5 i2c\_instance\_t Struct Reference

Structure storing PAL instance information.

```
#include <platform/pal/inc/i2c_pal_mapping.h>
```

#### Data Fields

- i2c\_inst\_type\_t [instType](#)
- uint32\_t [instIdx](#)

#### 17.5.1 Detailed Description

Structure storing PAL instance information.

This structure is used for storing PAL instance information. Implements : `i2c_instance_t_Class`

Definition at line 81 of file `i2c_pal_mapping.h`.

#### 17.5.2 Field Documentation

##### 17.5.2.1 uint32\_t instIdx

Instance index of the peripheral over which the PAL is used

Definition at line 83 of file `i2c_pal_mapping.h`.

##### 17.5.2.2 i2c\_inst\_type\_t instType

Peripheral over which the PAL is used

Definition at line 82 of file `i2c_pal_mapping.h`.

The documentation for this struct was generated from the following file:

- platform/pal/inc/i2c\_pal\_mapping.h

## 17.6 i2s\_instance\_t Struct Reference

Structure storing PAL instance information.

```
#include <platform/pal/inc/i2s_pal_mapping.h>
```

### Data Fields

- i2s\_inst\_type\_t [instType](#)
- uint32\_t [instIdx](#)

### 17.6.1 Detailed Description

Structure storing PAL instance information.

This structure is used for storing PAL instance information.

Definition at line 60 of file i2s\_pal\_mapping.h.

### 17.6.2 Field Documentation

#### 17.6.2.1 uint32\_t instIdx

Instance index of the peripheral over which the PAL is used

Definition at line 62 of file i2s\_pal\_mapping.h.

#### 17.6.2.2 i2s\_inst\_type\_t instType

Peripheral over which the PAL is used

Definition at line 61 of file i2s\_pal\_mapping.h.

The documentation for this struct was generated from the following file:

- platform/pal/inc/i2s\_pal\_mapping.h

## 17.7 ic\_instance\_t Struct Reference

Structure storing PAL instance information.

```
#include <platform/pal/inc/ic_pal_mapping.h>
```

### Data Fields

- ic\_inst\_type\_t [instType](#)
- uint32\_t [instIdx](#)

### 17.7.1 Detailed Description

Structure storing PAL instance information.

This structure is used for storing PAL instance information. Implements : ic\_instance\_t\_Class

Definition at line 132 of file ic\_pal\_mapping.h.

### 17.7.2 Field Documentation

#### 17.7.2.1 uint32\_t instIdx

Instance index of the peripheral over which the PAL is used

Definition at line 134 of file ic\_pal\_mapping.h.

#### 17.7.2.2 ic\_inst\_type\_t instType

Peripheral over which the PAL is used

Definition at line 133 of file ic\_pal\_mapping.h.

The documentation for this struct was generated from the following file:

- platform/pal/inc/ic\_pal\_mapping.h

## 17.8 lin\_product\_id\_t Struct Reference

Product id structure Implements : lin\_product\_id\_t\_Class.

```
#include <middleware/lin/include/lin_types.h>
```

### Data Fields

- l\_u16 [supplier\\_id](#)
- l\_u16 [function\\_id](#)
- l\_u8 [variant](#)

### 17.8.1 Detailed Description

Product id structure Implements : lin\_product\_id\_t\_Class.

Definition at line 54 of file lin\_types.h.

### 17.8.2 Field Documentation

#### 17.8.2.1 l\_u16 function\_id

Function ID

Definition at line 57 of file lin\_types.h.

#### 17.8.2.2 l\_u16 supplier\_id

Supplier ID

Definition at line 56 of file lin\_types.h.

#### 17.8.2.3 l\_u8 variant

Variant value

Definition at line 58 of file lin\_types.h.

The documentation for this struct was generated from the following file:

- middleware/lin/include/lin\_types.h

## 17.9 mpu\_instance\_t Struct Reference

Structure storing PAL instance information.

```
#include <platform/pal/inc/mpu_pal_mapping.h>
```

### Data Fields

- [mpu\\_inst\\_type\\_t instType](#)
- [uint32\\_t instIdx](#)

### 17.9.1 Detailed Description

Structure storing PAL instance information.

This structure is used for storing PAL instance information. Implements : `mpu_instance_t_Class`

Definition at line 68 of file `mpu_pal_mapping.h`.

### 17.9.2 Field Documentation

#### 17.9.2.1 `uint32_t instIdx`

Instance index of the peripheral over which the PAL is used

Definition at line 71 of file `mpu_pal_mapping.h`.

#### 17.9.2.2 `mpu_inst_type_t instType`

Peripheral over which the PAL is used

Definition at line 70 of file `mpu_pal_mapping.h`.

The documentation for this struct was generated from the following file:

- `platform/pal/inc/mpu_pal_mapping.h`

## 17.10 oc\_instance\_t Struct Reference

Structure storing PAL instance information.

```
#include <platform/pal/inc/oc_pal_mapping.h>
```

### Data Fields

- [oc\\_inst\\_type\\_t instType](#)
- [uint32\\_t instIdx](#)

### 17.10.1 Detailed Description

Structure storing PAL instance information.

This structure is used for storing PAL instance information. Implements : `oc_instance_t_Class`

Definition at line 110 of file `oc_pal_mapping.h`.

### 17.10.2 Field Documentation

#### 17.10.2.1 uint32\_t instIdx

Instance index of the peripheral over which the PAL is used

Definition at line 112 of file oc\_pal\_mapping.h.

#### 17.10.2.2 oc\_inst\_type\_t instType

Peripheral over which the PAL is used

Definition at line 111 of file oc\_pal\_mapping.h.

The documentation for this struct was generated from the following file:

- platform/pal/inc/oc\_pal\_mapping.h

## 17.11 oc\_pal\_state\_t Struct Reference

The internal context structure.

### 17.11.1 Detailed Description

The internal context structure.

This structure is used by the driver for its internal logic. It must be provided by the application through the [OC\\_Init\(\)](#) function, then it cannot be freed until the driver is de-initialized using [OC\\_Deinit\(\)](#). The application should make no assumptions about the content of this structure.

Definition at line 97 of file oc\_pal.c.

The documentation for this struct was generated from the following file:

- platform/pal/src/oc/oc\_pal.c

## 17.12 pwm\_instance\_t Struct Reference

Structure storing PAL instance information.

```
#include <platform/pal/inc/pwm_pal_mapping.h>
```

### Data Fields

- pwm\_inst\_type\_t [instType](#)
- uint32\_t [instIdx](#)

### 17.12.1 Detailed Description

Structure storing PAL instance information.

This structure is used for storing PAL instance information. Implements : `pwm_instance_t_Class`

Definition at line 46 of file pwm\_pal\_mapping.h.

### 17.12.2 Field Documentation

#### 17.12.2.1 uint32\_t instIdx

Instance index of the peripheral over which the PAL is used

Definition at line 48 of file pwm\_pal\_mapping.h.

#### 17.12.2.2 pwm\_inst\_type\_t instType

Peripheral over which the PAL is used

Definition at line 47 of file pwm\_pal\_mapping.h.

The documentation for this struct was generated from the following file:

- platform/pal/inc/pwm\_pal\_mapping.h

### 17.13 spi\_instance\_t Struct Reference

Structure storing PAL instance information.

```
#include <platform/pal/inc/spi_pal_mapping.h>
```

#### Data Fields

- spi\_inst\_type\_t [instType](#)
- uint32\_t [instIdx](#)

#### 17.13.1 Detailed Description

Structure storing PAL instance information.

This structure is used for storing PAL instance information. Implements : spi\_instance\_t\_Class

Definition at line 50 of file spi\_pal\_mapping.h.

#### 17.13.2 Field Documentation

##### 17.13.2.1 uint32\_t instIdx

Instance index of the peripheral over which the PAL is used

Definition at line 52 of file spi\_pal\_mapping.h.

##### 17.13.2.2 spi\_inst\_type\_t instType

Peripheral over which the PAL is used

Definition at line 51 of file spi\_pal\_mapping.h.

The documentation for this struct was generated from the following file:

- platform/pal/inc/spi\_pal\_mapping.h

### 17.14 timer\_chan\_state\_t Struct Reference

Runtime state of the Timer channel.

#### 17.14.1 Detailed Description

Runtime state of the Timer channel.

This structure is used by the driver for its internal logic. The application should make no assumptions about the content of this structure.

Definition at line 77 of file timing\_pal.c.

The documentation for this struct was generated from the following file:

- platform/pal/src/timing/timing\_pal.c

### 17.15 timing\_instance\_t Struct Reference

Structure storing PAL instance information.

```
#include <platform/pal/inc/timing_pal_mapping.h>
```

#### Data Fields

- timing\_inst\_type\_t [instType](#)
- uint32\_t [instIdx](#)

#### 17.15.1 Detailed Description

Structure storing PAL instance information.

This structure is used for storing PAL instance information. Implements : timing\_instance\_t\_Class

Definition at line 88 of file timing\_pal\_mapping.h.

#### 17.15.2 Field Documentation

##### 17.15.2.1 uint32\_t instIdx

Instance index of the peripheral over which the PAL is used

Definition at line 90 of file timing\_pal\_mapping.h.

##### 17.15.2.2 timing\_inst\_type\_t instType

Peripheral over which the PAL is used

Definition at line 89 of file timing\_pal\_mapping.h.

The documentation for this struct was generated from the following file:

- platform/pal/inc/timing\_pal\_mapping.h

### 17.16 uart\_instance\_t Struct Reference

Structure storing PAL instance information.

```
#include <platform/pal/inc/uart_pal_mapping.h>
```

#### Data Fields

- uart\_inst\_type\_t [instType](#)
- uint32\_t [instIdx](#)

### 17.16.1 Detailed Description

Structure storing PAL instance information.

This structure is used for storing PAL instance information. Implements : `uart_instance_t_Class`

Definition at line 63 of file `uart_pal_mapping.h`.

### 17.16.2 Field Documentation

#### 17.16.2.1 `uint32_t instIdx`

Instance index of the peripheral over which the PAL is used

Definition at line 66 of file `uart_pal_mapping.h`.

#### 17.16.2.2 `uart_inst_type_t instType`

Peripheral over which the PAL is used

Definition at line 65 of file `uart_pal_mapping.h`.

The documentation for this struct was generated from the following file:

- `platform/pal/inc/uart_pal_mapping.h`

## 17.17 `wdg_instance_t` Struct Reference

Structure storing PAL instance information.

```
#include <platform/pal/inc/wdg_pal_mapping.h>
```

### Data Fields

- [`wdg\_inst\_type\_t instType`](#)
- `uint32_t instIdx`

### 17.17.1 Detailed Description

Structure storing PAL instance information.

This structure is used for storing PAL instance information. Implements : `wdg_instance_t_Class`

Definition at line 63 of file `wdg_pal_mapping.h`.

### 17.17.2 Field Documentation

#### 17.17.2.1 `uint32_t instIdx`

Instance index of the peripheral over which the PAL is used

Definition at line 65 of file `wdg_pal_mapping.h`.

#### 17.17.2.2 `wdg_inst_type_t instType`

Peripheral over which the PAL is used

Definition at line 64 of file `wdg_pal_mapping.h`.

The documentation for this struct was generated from the following file:



- platform/pal/inc/wdg\_pal\_mapping.h



## Index

- 00. Library internals, [139](#)
  - [g\\_drivers](#), [149](#)
  - [UJA1131HW3V3](#), [148](#)
  - [UJA1131HW5V0](#), [148](#)
  - [UJA1131HWFD3V0](#), [148](#)
  - [UJA1131HWFD3V4](#), [148](#)
  - [UJA1131HWFD5V0](#), [148](#)
  - [UJA1131HWFD5V4](#), [148](#)
  - [UJA1132HW3V3](#), [148](#)
  - [UJA1132HW5V0](#), [148](#)
  - [UJA1132HWFD3V0](#), [148](#)
  - [UJA1132HWFD3V4](#), [148](#)
  - [UJA1132HWFD5V0](#), [148](#)
  - [UJA1132HWFD5V4](#), [148](#)
  - [UJA1135HW3V3](#), [148](#)
  - [UJA1135HW5V0](#), [148](#)
  - [UJA1136HW3V3](#), [148](#)
  - [UJA1136HW5V0](#), [148](#)
  - [UJA113X\\_Init](#), [148](#)
  - [UJA113X\\_RegisterDevice](#), [148](#)
  - [UJA113X\\_SbcTransfer](#), [149](#)
  - [uja113x\\_chip\\_t](#), [148](#)
- 01. CAN & CAN partial networking configuration, [150](#)
  - [CDR\\_DATA\\_RATE\\_100](#), [156](#)
  - [CDR\\_DATA\\_RATE\\_1000](#), [156](#)
  - [CDR\\_DATA\\_RATE\\_125](#), [156](#)
  - [CDR\\_DATA\\_RATE\\_250](#), [156](#)
  - [CDR\\_DATA\\_RATE\\_50](#), [156](#)
  - [CDR\\_DATA\\_RATE\\_500a](#), [156](#)
  - [CDR\\_DATA\\_RATE\\_500b](#), [156](#)
  - [CDR\\_DATA\\_RATE\\_500c](#), [156](#)
  - [CMC\\_TRANSCEIVER\\_ACTIVE](#), [155](#)
  - [CMC\\_TRANSCEIVER\\_ACTIVE2](#), [155](#)
  - [CMC\\_TRANSCEIVER\\_LISTEN](#), [155](#)
  - [CMC\\_TRANSCEIVER\\_OFF](#), [155](#)
  - [CSC\\_NONE](#), [155](#)
  - [CSC\\_OVERVOLTAGE](#), [155](#)
  - [CSC\\_UNDRV\\_OVERV](#), [155](#)
  - [CSC\\_UNDRVOLTAGE](#), [155](#)
  - [DLC\\_0](#), [156](#)
  - [DLC\\_1](#), [156](#)
  - [DLC\\_2](#), [156](#)
  - [DLC\\_3](#), [156](#)
  - [DLC\\_4](#), [156](#)
  - [DLC\\_5](#), [156](#)
  - [DLC\\_6](#), [156](#)
  - [DLC\\_7](#), [156](#)
  - [DLC\\_8](#), [156](#)
  - [UJA113X\\_SetCan](#), [156](#)
  - [UJA113X\\_SetCanPartNet](#), [157](#)
  - [UJA113X\\_SetDataMasks](#), [157](#)
  - [uja113x\\_canc\\_cfdc\\_t](#), [154](#)
  - [uja113x\\_canc\\_cmc\\_t](#), [155](#)
  - [uja113x\\_canc\\_cpnc\\_t](#), [154](#)
  - [uja113x\\_canc\\_csc\\_t](#), [155](#)
  - [uja113x\\_dr\\_cdr\\_t](#), [155](#)
  - [uja113x\\_fc\\_dlc\\_t](#), [156](#)
  - [uja113x\\_fc\\_ide\\_t](#), [154](#)
  - [uja113x\\_fc\\_pndm\\_t](#), [154](#)
  - [uja113x\\_id\\_id\\_t](#), [155](#)
  - [uja113x\\_idm\\_idm\\_t](#), [155](#)
- 01. Mode handling, [159](#)
  - [MC\\_NORMAL](#), [159](#)
  - [MC\\_SLEEP](#), [159](#)
  - [MC\\_STANDBY](#), [159](#)
  - [UJA113X\\_SetMode](#), [160](#)
  - [uja113x\\_mc\\_mc\\_t](#), [159](#)
- 01. Primary control, [161](#)
- 01. Regulator configuration, [162](#)
  - [UJA113X\\_SetRegulator](#), [164](#)
  - [uja113x\\_rc\\_v1rtc\\_t](#), [163](#)
  - [uja113x\\_rc\\_v2c\\_t](#), [163](#)
  - [uja113x\\_rc\\_v2sc\\_t](#), [163](#)
  - [V1RTC\\_60\\_NOMINAL\\_OUT](#), [163](#)
  - [V1RTC\\_70\\_NOMINAL\\_OUT](#), [163](#)
  - [V1RTC\\_80\\_NOMINAL\\_OUT](#), [163](#)
  - [V1RTC\\_90\\_NOMINAL\\_OUT](#), [163](#)
  - [V2C\\_OFF](#), [163](#)
  - [V2C\\_ON\\_N](#), [163](#)
  - [V2C\\_ON\\_NSR](#), [163](#)
  - [V2C\\_ON\\_NSSRF](#), [163](#)
  - [V2SC\\_FULL](#), [164](#)
  - [V2SC\\_NONE](#), [163](#)
  - [V2SC\\_OVERVOLTAGE](#), [163](#)
  - [V2SC\\_UNDRVOLTAGE](#), [163](#)
- 02. Battery monitor configuration, [165](#)
  - [IONOLTC\\_10\\_MA](#), [168](#)
  - [IONOLTC\\_20\\_MA](#), [168](#)
  - [IONOLTC\\_2\\_MA](#), [167](#)
  - [IONOLTC\\_5\\_MA](#), [167](#)
  - [IONSCTC\\_30\\_MA](#), [168](#)
  - [IONSCTC\\_45\\_MA](#), [168](#)
  - [IONSCTC\\_60\\_MA](#), [168](#)
  - [IONSCTC\\_90\\_MA](#), [168](#)
  - [UJA113X\\_SetBatteryMonitor](#), [168](#)
  - [uja113x\\_bmetsc\\_bmsc\\_t](#), [167](#)
  - [uja113x\\_bmhc\\_bmhc\\_t](#), [167](#)
  - [uja113x\\_bmhc\\_bmhc\\_t](#), [167](#)
  - [uja113x\\_bmotc\\_bmotc\\_t](#), [167](#)
  - [uja113x\\_bmutc\\_bmutc\\_t](#), [167](#)
  - [uja113x\\_boldt\\_ionoltc\\_t](#), [167](#)
  - [uja113x\\_bscdt\\_ionsctc\\_t](#), [168](#)
- 02. LIN configuration, [169](#)
  - [LMC1\\_TRANSCEIVER\\_LISTEN](#), [170](#)
  - [LMC1\\_TRANSCEIVER\\_OFF](#), [170](#)
  - [LMC1\\_TRANSCEIVER\\_ON](#), [170](#)
  - [LMC1\\_TRANSCEIVER\\_ON2](#), [170](#)
  - [LMC2\\_TRANSCEIVER\\_LISTEN](#), [171](#)
  - [LMC2\\_TRANSCEIVER\\_OFF](#), [171](#)
  - [LMC2\\_TRANSCEIVER\\_ON](#), [171](#)

- LMC2\_TRANSCEIVER\_ON2, [171](#)
- LSC1\_SLOPE\_ACTIVE, [171](#)
- LSC1\_SLOPE\_ACTIVE2, [171](#)
- LSC1\_SLOPE\_ACTIVE\_TXDL\_TIMEOUT, [171](#)
- LSC2\_SLOPE\_ACTIVE, [171](#)
- LSC2\_SLOPE\_ACTIVE2, [171](#)
- LSC2\_SLOPE\_ACTIVE\_TXDL\_TIMEOUT, [171](#)
- UJA113X\_SetLin, [171](#)
- uja113x\_linc\_lmc1\_t, [170](#)
- uja113x\_linc\_lmc2\_t, [170](#)
- uja113x\_linc\_lsc1\_t, [171](#)
- uja113x\_linc\_lsc2\_t, [171](#)
- 02. Supply control, [173](#)
- 02. Watchdog, [174](#)
  - NWP\_PERIOD\_1024, [176](#)
  - NWP\_PERIOD\_128, [176](#)
  - NWP\_PERIOD\_16, [176](#)
  - NWP\_PERIOD\_256, [176](#)
  - NWP\_PERIOD\_32, [176](#)
  - NWP\_PERIOD\_4096, [176](#)
  - NWP\_PERIOD\_64, [176](#)
  - NWP\_PERIOD\_8, [176](#)
  - UJA113X\_SetWatchdog, [176](#)
  - uja113x\_wtdc\_nwp\_t, [175](#)
  - uja113x\_wtdc\_wmc\_t, [176](#)
  - WMC\_AUTONOMOUS, [176](#)
  - WMC\_TIMEOUT, [176](#)
  - WMC\_WINDOW, [176](#)
- 03. Fail-safe configuration, [178](#)
  - ENC\_HIGH\_N, [180](#)
  - ENC\_HIGH\_NRS, [180](#)
  - ENC\_LOW, [180](#)
  - ENC\_TIMER, [180](#)
  - ENSC\_FULL, [180](#)
  - ENSC\_NONE, [180](#)
  - ENSC\_OVERVOLTAGE, [180](#)
  - ENSC\_UNDRVOLTAGE, [180](#)
  - UJA113X\_SetFailSafe, [181](#)
  - uja113x\_fsc\_enc\_t, [180](#)
  - uja113x\_fsc\_endc\_t, [180](#)
  - uja113x\_fsc\_ensc\_t, [180](#)
  - uja113x\_fsc\_lhc\_t, [180](#)
  - uja113x\_fsc\_rcc\_t, [180](#)
- 03. SMPS configuration, [182](#)
  - SMPSOC\_ON\_NSR, [183](#)
  - SMPSOC\_ON\_NSRS, [183](#)
  - SMPSOC\_PASS\_THROUGH\_NSS, [183](#)
  - SMPSOC\_50V, [183](#)
  - SMPSOC\_52V, [183](#)
  - SMPSOC\_54V, [183](#)
  - SMPSOC\_56V, [183](#)
  - SMPSOC\_58V, [184](#)
  - SMPSOC\_60V, [184](#)
  - SMPSOC\_62V, [184](#)
  - SMPSOC\_64V, [184](#)
  - SMPSOC\_66V, [184](#)
  - SMPSOC\_68V, [184](#)
  - SMPSOC\_70V, [184](#)
  - SMPSOC\_72V, [184](#)
  - SMPSOC\_74V, [184](#)
  - SMPSOC\_76V, [184](#)
  - SMPSOC\_78V, [184](#)
  - SMPSOC\_80V, [184](#)
  - UJA113X\_SetSmpls, [184](#)
  - uja113x\_smpsc\_smpsc\_t, [183](#)
  - uja113x\_smpsc\_smpsotc\_t, [183](#)
  - uja113x\_smpsovc\_smpsoc\_t, [183](#)
- 03. Transceiver control, [185](#)
- 04. HVIO bank 0 & 1 control, [186](#)
  - IONAC\_DEACTIVATED, [190](#)
  - IONAC\_ENABLED, [190](#)
  - IONAC\_HVIO, [190](#)
  - IONAC\_HVIO\_INV, [190](#)
  - IONAC\_TMR\_1, [190](#)
  - IONAC\_TMR\_2, [190](#)
  - IONAC\_TMR\_3, [190](#)
  - IONAC\_TMR\_4, [190](#)
  - IONCC\_HS, [190](#)
  - IONCC\_HS\_SLOPE, [190](#)
  - IONCC\_HS\_WUP, [190](#)
  - IONCC\_HS\_WUP\_SLOPE, [190](#)
  - IONCC\_LS, [190](#)
  - IONCC\_LS\_SLOPE, [190](#)
  - IONCC\_OFF, [190](#)
  - IONCC\_WUP, [190](#)
  - IONSC\_FULL, [190](#)
  - IONSC\_NONE, [190](#)
  - IONSC\_OVERVOLTAGE, [190](#)
  - IONSC\_UNDRVOLTAGE, [190](#)
  - UJA113X\_SetBankOpenLoadThr, [190](#)
  - UJA113X\_SetBankShortCircThr, [191](#)
  - UJA113X\_SetBankWupThr, [191](#)
  - UJA113X\_SetHvio, [192](#)
  - uja113x\_bwtc\_bnwtc\_t, [189](#)
  - uja113x\_hvion\_ionac\_t, [189](#)
  - uja113x\_hvion\_ioncc\_t, [190](#)
  - uja113x\_hvion\_ionsc\_t, [190](#)
- 04. Lock control, [193](#)
  - UJA113X\_SetLock, [195](#)
  - uja113x\_lckc\_lkxc\_t, [195](#)
- 05. Timer control, [196](#)
  - TXMC\_PWM, [198](#)
  - TXMC\_TIM, [198](#)
  - TXMC\_TRIG\_END, [198](#)
  - TXMC\_TRIG\_START, [198](#)
  - TXPC\_1000MS\_PERIOD, [198](#)
  - TXPC\_100MS\_PERIOD, [198](#)
  - TXPC\_2000MS\_PERIOD, [198](#)
  - TXPC\_200MS\_PERIOD, [198](#)
  - TXPC\_20MS\_PERIOD, [198](#)
  - TXPC\_30MS\_PERIOD, [198](#)
  - TXPC\_4000MS\_PERIOD, [198](#)
  - TXPC\_400MS\_PERIOD, [198](#)
  - TXPC\_4MS\_PERIOD, [198](#)
  - TXPC\_50MS\_PERIOD, [198](#)
  - TXPC\_800MS\_PERIOD, [198](#)

- TXPC\_8MS\_PERIOD, [198](#)
- UJA113X\_SetTimer, [198](#)
- uja113x\_timxc\_txmc\_t, [197](#)
- uja113x\_timxc\_txpc\_t, [198](#)
- uja113x\_timxdcc\_txdcc\_t, [197](#)
- 06. Interrupts, [200](#)
  - UJA113X\_SetBankFailInterrupts, [200](#)
  - UJA113X\_SetBankWupInterrupts, [201](#)
  - UJA113X\_SetSupplyInterrupts, [201](#)
  - UJA113X\_SetSystemInterrupts, [202](#)
  - UJA113X\_SetTransceiverInterrupts, [202](#)
- 07. MTPNV and ID, [204](#)
  - UJA113X\_ComputeCrc8, [209](#)
  - UJA113X\_SetMtpnv, [209](#)
- ADC Driver, [211](#)
  - ADC\_AVERAGE\_16, [220](#)
  - ADC\_AVERAGE\_32, [220](#)
  - ADC\_AVERAGE\_4, [220](#)
  - ADC\_AVERAGE\_8, [220](#)
  - ADC\_CLK\_ALT\_1, [220](#)
  - ADC\_CLK\_ALT\_2, [220](#)
  - ADC\_CLK\_ALT\_3, [220](#)
  - ADC\_CLK\_ALT\_4, [220](#)
  - ADC\_CLK\_DIVIDE\_1, [220](#)
  - ADC\_CLK\_DIVIDE\_2, [220](#)
  - ADC\_CLK\_DIVIDE\_4, [220](#)
  - ADC\_CLK\_DIVIDE\_8, [220](#)
  - ADC\_DRV\_AutoCalibration, [223](#)
  - ADC\_DRV\_ClearLatchedTriggers, [223](#)
  - ADC\_DRV\_ClearTriggerErrors, [223](#)
  - ADC\_DRV\_ConfigChan, [223](#)
  - ADC\_DRV\_ConfigConverter, [224](#)
  - ADC\_DRV\_ConfigHwAverage, [224](#)
  - ADC\_DRV\_ConfigHwCompare, [224](#)
  - ADC\_DRV\_ConfigUserCalibration, [224](#)
  - ADC\_DRV\_GetChanConfig, [224](#)
  - ADC\_DRV\_GetChanResult, [225](#)
  - ADC\_DRV\_GetConvCompleteFlag, [225](#)
  - ADC\_DRV\_GetConverterConfig, [225](#)
  - ADC\_DRV\_GetHwAverageConfig, [225](#)
  - ADC\_DRV\_GetHwCompareConfig, [226](#)
  - ADC\_DRV\_GetInterruptNumber, [226](#)
  - ADC\_DRV\_GetTriggerErrorFlags, [226](#)
  - ADC\_DRV\_GetUserCalibration, [226](#)
  - ADC\_DRV\_InitChanStruct, [226](#)
  - ADC\_DRV\_InitConverterStruct, [227](#)
  - ADC\_DRV\_InitHwAverageStruct, [227](#)
  - ADC\_DRV\_InitHwCompareStruct, [227](#)
  - ADC\_DRV\_InitUserCalibrationStruct, [227](#)
  - ADC\_DRV\_Reset, [228](#)
  - ADC\_DRV\_SetSwPretrigger, [228](#)
  - ADC\_DRV\_WaitConvDone, [228](#)
  - ADC\_INPUTCHAN\_BANDGAP, [221](#)
  - ADC\_INPUTCHAN\_DISABLED, [221](#)
  - ADC\_INPUTCHAN\_EXT0, [220](#)
  - ADC\_INPUTCHAN\_EXT1, [220](#)
  - ADC\_INPUTCHAN\_EXT10, [221](#)
  - ADC\_INPUTCHAN\_EXT11, [221](#)
  - ADC\_INPUTCHAN\_EXT12, [221](#)
  - ADC\_INPUTCHAN\_EXT13, [221](#)
  - ADC\_INPUTCHAN\_EXT14, [221](#)
  - ADC\_INPUTCHAN\_EXT3, [220](#)
  - ADC\_INPUTCHAN\_EXT4, [221](#)
  - ADC\_INPUTCHAN\_EXT5, [221](#)
  - ADC\_INPUTCHAN\_EXT6, [221](#)
  - ADC\_INPUTCHAN\_EXT7, [221](#)
  - ADC\_INPUTCHAN\_EXT9, [221](#)
  - ADC\_INPUTCHAN\_INT0, [221](#)
  - ADC\_INPUTCHAN\_INT1, [221](#)
  - ADC\_INPUTCHAN\_INT2, [221](#)
  - ADC\_INPUTCHAN\_INT3, [221](#)
  - ADC\_INPUTCHAN\_SUPPLY\_VDD, [221](#)
  - ADC\_INPUTCHAN\_SUPPLY\_VDD\_3V, [221](#)
  - ADC\_INPUTCHAN\_SUPPLY\_VDD\_FLASH\_3V, [221](#)
  - ADC\_INPUTCHAN\_SUPPLY\_VDD\_LV, [221](#)
  - ADC\_INPUTCHAN\_SUPPLY\_VDDA, [221](#)
  - ADC\_INPUTCHAN\_SUPPLY\_VREFH, [221](#)
  - ADC\_INPUTCHAN\_TEMP, [221](#)
  - ADC\_INPUTCHAN\_VREFSH, [221](#)
  - ADC\_INPUTCHAN\_VREFSL, [221](#)
  - ADC\_LATCH\_CLEAR\_FORCE, [221](#)
  - ADC\_LATCH\_CLEAR\_WAIT, [221](#)
  - ADC\_PRETRIGGER\_SEL\_PDB, [221](#)
  - ADC\_PRETRIGGER\_SEL\_SW, [221](#)
  - ADC\_PRETRIGGER\_SEL\_TRGMUX, [221](#)
  - ADC\_RESOLUTION\_10BIT, [222](#)
  - ADC\_RESOLUTION\_12BIT, [222](#)
  - ADC\_RESOLUTION\_8BIT, [222](#)
  - ADC\_SW\_PRETRIGGER\_0, [222](#)
  - ADC\_SW\_PRETRIGGER\_1, [222](#)
  - ADC\_SW\_PRETRIGGER\_2, [222](#)
  - ADC\_SW\_PRETRIGGER\_3, [222](#)
  - ADC\_SW\_PRETRIGGER\_DISABLED, [222](#)
  - ADC\_TRIGGER\_HARDWARE, [222](#)
  - ADC\_TRIGGER\_SEL\_PDB, [222](#)
  - ADC\_TRIGGER\_SEL\_TRGMUX, [222](#)
  - ADC\_TRIGGER\_SOFTWARE, [222](#)
  - ADC\_VOLTAGEREF\_VALT, [223](#)
  - ADC\_VOLTAGEREF\_VREF, [223](#)
  - adc\_average\_t, [219](#)
  - adc\_clk\_divide\_t, [220](#)
  - adc\_input\_clock\_t, [220](#)
  - adc\_inputchannel\_t, [220](#)
  - adc\_latch\_clear\_t, [221](#)
  - adc\_pretrigger\_sel\_t, [221](#)
  - adc\_resolution\_t, [221](#)
  - adc\_sw\_pretrigger\_t, [222](#)
  - adc\_trigger\_sel\_t, [222](#)
  - adc\_trigger\_t, [222](#)
  - adc\_voltage\_reference\_t, [222](#)
- ADC\_AVERAGE\_16
  - ADC Driver, [220](#)
- ADC\_AVERAGE\_32
  - ADC Driver, [220](#)
- ADC\_AVERAGE\_4

- ADC Driver, [220](#)
- ADC\_AVERAGE\_8
  - ADC Driver, [220](#)
- ADC\_CLK\_ALT\_1
  - ADC Driver, [220](#)
- ADC\_CLK\_ALT\_2
  - ADC Driver, [220](#)
- ADC\_CLK\_ALT\_3
  - ADC Driver, [220](#)
- ADC\_CLK\_ALT\_4
  - ADC Driver, [220](#)
- ADC\_CLK\_DIVIDE\_1
  - ADC Driver, [220](#)
- ADC\_CLK\_DIVIDE\_2
  - ADC Driver, [220](#)
- ADC\_CLK\_DIVIDE\_4
  - ADC Driver, [220](#)
- ADC\_CLK\_DIVIDE\_8
  - ADC Driver, [220](#)
- ADC\_DELAY\_TYPE\_GROUP\_DELAY
  - Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), [237](#)
- ADC\_DELAY\_TYPE\_INDIVIDUAL\_DELAY
  - Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), [237](#)
- ADC\_DELAY\_TYPE\_NO\_DELAY
  - Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), [237](#)
- ADC\_DRV\_AutoCalibration
  - ADC Driver, [223](#)
- ADC\_DRV\_ClearLatchedTriggers
  - ADC Driver, [223](#)
- ADC\_DRV\_ClearTriggerErrors
  - ADC Driver, [223](#)
- ADC\_DRV\_ConfigChan
  - ADC Driver, [223](#)
- ADC\_DRV\_ConfigConverter
  - ADC Driver, [224](#)
- ADC\_DRV\_ConfigHwAverage
  - ADC Driver, [224](#)
- ADC\_DRV\_ConfigHwCompare
  - ADC Driver, [224](#)
- ADC\_DRV\_ConfigUserCalibration
  - ADC Driver, [224](#)
- ADC\_DRV\_GetChanConfig
  - ADC Driver, [224](#)
- ADC\_DRV\_GetChanResult
  - ADC Driver, [225](#)
- ADC\_DRV\_GetConvCompleteFlag
  - ADC Driver, [225](#)
- ADC\_DRV\_GetConverterConfig
  - ADC Driver, [225](#)
- ADC\_DRV\_GetHwAverageConfig
  - ADC Driver, [225](#)
- ADC\_DRV\_GetHwCompareConfig
  - ADC Driver, [226](#)
- ADC\_DRV\_GetInterruptNumber
  - ADC Driver, [226](#)
- ADC\_DRV\_GetTriggerErrorFlags
  - ADC Driver, [226](#)
- ADC\_DRV\_GetUserCalibration
  - ADC Driver, [226](#)
- ADC\_DRV\_InitChanStruct
  - ADC Driver, [226](#)
- ADC\_DRV\_InitConverterStruct
  - ADC Driver, [227](#)
- ADC\_DRV\_InitHwAverageStruct
  - ADC Driver, [227](#)
- ADC\_DRV\_InitHwCompareStruct
  - ADC Driver, [227](#)
- ADC\_DRV\_InitUserCalibrationStruct
  - ADC Driver, [227](#)
- ADC\_DRV\_Reset
  - ADC Driver, [228](#)
- ADC\_DRV\_SetSwPretrigger
  - ADC Driver, [228](#)
- ADC\_DRV\_WaitConvDone
  - ADC Driver, [228](#)
- ADC\_Deinit
  - Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), [237](#)
- ADC\_DisableHardwareTrigger
  - Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), [239](#)
- ADC\_DisableNotification
  - Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), [239](#)
- ADC\_EnableHardwareTrigger
  - Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), [239](#)
- ADC\_EnableNotification
  - Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), [240](#)
- ADC\_INPUTCHAN\_BANDGAP
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_DISABLED
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_EXT0
  - ADC Driver, [220](#)
- ADC\_INPUTCHAN\_EXT1
  - ADC Driver, [220](#)
- ADC\_INPUTCHAN\_EXT10
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_EXT11
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_EXT12
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_EXT13
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_EXT14
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_EXT3
  - ADC Driver, [220](#)
- ADC\_INPUTCHAN\_EXT4
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_EXT5

- ADC Driver, [221](#)
- ADC\_INPUTCHAN\_EXT6
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_EXT7
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_EXT9
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_INT0
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_INT1
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_INT2
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_INT3
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_SUPPLY\_VDD
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_SUPPLY\_VDD\_3V
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_SUPPLY\_VDD\_FLASH\_3V
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_SUPPLY\_VDD\_LV
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_SUPPLY\_VDDA
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_SUPPLY\_VREFH
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_TEMP
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_VREFSH
  - ADC Driver, [221](#)
- ADC\_INPUTCHAN\_VREFSL
  - ADC Driver, [221](#)
- ADC\_Init
  - Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), [240](#)
- ADC\_LATCH\_CLEAR\_FORCE
  - ADC Driver, [221](#)
- ADC\_LATCH\_CLEAR\_WAIT
  - ADC Driver, [221](#)
- ADC\_PRETRIGGER\_SEL\_PDB
  - ADC Driver, [221](#)
- ADC\_PRETRIGGER\_SEL\_SW
  - ADC Driver, [221](#)
- ADC\_PRETRIGGER\_SEL\_TRGMUX
  - ADC Driver, [221](#)
- ADC\_RESOLUTION\_10BIT
  - ADC Driver, [222](#)
- ADC\_RESOLUTION\_12BIT
  - ADC Driver, [222](#)
- ADC\_RESOLUTION\_8BIT
  - ADC Driver, [222](#)
- ADC\_SW\_PRETRIGGER\_0
  - ADC Driver, [222](#)
- ADC\_SW\_PRETRIGGER\_1
  - ADC Driver, [222](#)
- ADC\_SW\_PRETRIGGER\_2
  - ADC Driver, [222](#)
- ADC\_SW\_PRETRIGGER\_3
  - ADC Driver, [222](#)
- ADC\_SW\_PRETRIGGER\_DISABLED
  - ADC Driver, [222](#)
- ADC\_StartGroupConversion
  - Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), [240](#)
- ADC\_StopGroupConversion
  - Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), [241](#)
- ADC\_TRIGGER\_HARDWARE
  - ADC Driver, [222](#)
- ADC\_TRIGGER\_SEL\_PDB
  - ADC Driver, [222](#)
- ADC\_TRIGGER\_SEL\_TRGMUX
  - ADC Driver, [222](#)
- ADC\_TRIGGER\_SOFTWARE
  - ADC Driver, [222](#)
- ADC\_VOLTAGEREF\_VALT
  - ADC Driver, [223](#)
- ADC\_VOLTAGEREF\_VREF
  - ADC Driver, [223](#)
- ALL\_MODES
  - Clock Manager Driver, [305](#)
- ATTR
  - edma\_software\_tcd\_t, [382](#)
- accessCtr
  - mpu\_access\_err\_info\_t, [790](#)
  - mpu\_error\_info\_t, [801](#)
- accessRight
  - mpu\_master\_access\_permission\_t, [802](#)
  - mpu\_master\_access\_right\_t, [790](#)
- accessType
  - mpu\_access\_err\_info\_t, [790](#)
  - mpu\_error\_info\_t, [801](#)
- active\_schedule\_id
  - lin\_master\_data\_t, [768](#)
- adc\_average\_config\_t, [218](#)
  - hwAverage, [218](#)
  - hwAvgEnable, [218](#)
- adc\_average\_t
  - ADC Driver, [219](#)
- adc\_calibration\_t, [219](#)
  - userGain, [219](#)
  - userOffset, [219](#)
- adc\_callback\_info\_t, [1081](#)
  - groupIndex, [1081](#)
  - resultBufferTail, [1081](#)
- adc\_chan\_config\_t, [218](#)
  - channel, [219](#)
  - interruptEnable, [219](#)
- adc\_clk\_divide\_t
  - ADC Driver, [220](#)
- adc\_compare\_config\_t, [217](#)
  - compVal1, [218](#)
  - compVal2, [218](#)
  - compareEnable, [218](#)
  - compareGreaterThanEnable, [218](#)

- compareRangeFuncEnable, 218
- adc\_config\_t, 235
  - extension, 236
  - groupConfigArray, 236
  - numGroups, 236
  - sampleTicks, 236
- adc\_converter\_config\_t, 216
  - clockDivide, 216
  - continuousConvEnable, 216
  - dmaEnable, 216
  - inputClock, 217
  - pretriggerSel, 217
  - resolution, 217
  - sampleTime, 217
  - supplyMonitoringEnable, 217
  - trigger, 217
  - triggerSel, 217
  - voltageRef, 217
- adc\_delay\_type\_t
  - Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), 237
- adc\_group\_config\_t, 234
  - callback, 234
  - callbackUserData, 234
  - continuousConvEn, 234
  - delayArray, 234
  - delayType, 235
  - hwTriggerSupport, 235
  - inputChannelArray, 235
  - numChannels, 235
  - numSetsResultBuffer, 235
  - resultBuffer, 235
  - triggerSource, 235
- adc\_input\_chan\_t
  - Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), 237
- adc\_input\_clock\_t
  - ADC Driver, 220
- adc\_inputchannel\_t
  - ADC Driver, 220
- adc\_instance\_t, 1081
  - instIdx, 1082
  - instType, 1082
- adc\_latch\_clear\_t
  - ADC Driver, 221
- adc\_pretrigger\_sel\_t
  - ADC Driver, 221
- adc\_resolution\_t
  - ADC Driver, 221
- adc\_sw\_pretrigger\_t
  - ADC Driver, 222
- adc\_trigger\_sel\_t
  - ADC Driver, 222
- adc\_trigger\_source\_t
  - Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), 237
- adc\_trigger\_t
  - ADC Driver, 222
- adc\_voltage\_reference\_t
  - ADC Driver, 222
- adcPreTriggerIdx
  - pdb\_adc\_pretrigger\_config\_t, 850
- addr
  - mpu\_access\_err\_info\_t, 790
  - mpu\_error\_info\_t, 801
- address
  - edma\_scatter\_gather\_list\_t, 378
- alarmCallback
  - rtc\_alarm\_config\_t, 897
- alarmIntEnable
  - rtc\_alarm\_config\_t, 897
- alarmTime
  - rtc\_alarm\_config\_t, 897
- alternateClock
  - scg\_clock\_mode\_config\_t, 294
- Analog to Digital Converter - Peripheral Abstraction Layer (ADC PAL), 229
  - ADC\_DELAY\_TYPE\_GROUP\_DELAY, 237
  - ADC\_DELAY\_TYPE\_INDIVIDUAL\_DELAY, 237
  - ADC\_DELAY\_TYPE\_NO\_DELAY, 237
  - ADC\_Deinit, 237
  - ADC\_DisableHardwareTrigger, 239
  - ADC\_DisableNotification, 239
  - ADC\_EnableHardwareTrigger, 239
  - ADC\_EnableNotification, 240
  - ADC\_Init, 240
  - ADC\_StartGroupConversion, 240
  - ADC\_StopGroupConversion, 241
  - adc\_delay\_type\_t, 237
  - adc\_input\_chan\_t, 237
  - adc\_trigger\_source\_t, 237
- assertLogic
  - ewm\_init\_config\_t, 408
  - extension\_ewm\_for\_wdg\_t, 1062
- associated\_uncond\_frame\_ptr
  - lin\_associate\_frame\_t, 759
- attributes
  - mpu\_access\_err\_info\_t, 790
  - mpu\_error\_info\_t, 801
- autoClearTrigger
  - ftm\_pwm\_sync\_t, 523
- autobaudEnable
  - lin\_user\_config\_t, 653
- Automotive Math and Motor Control Library, 242
- B0DRVS\_ADDRESS
  - Register address map, 918
- B0FIE\_ADDRESS
  - Register address map, 918
- B0FIS\_ADDRESS
  - Register address map, 918
- B0OLDTC\_ADDRESS
  - Register address map, 918
- B0SCDTC\_ADDRESS
  - Register address map, 918
- B0WUPIE\_ADDRESS
  - Register address map, 918



- B0WUPIS\_ADDRESS
  - Register address map, [918](#)
- B0WUPS\_ADDRESS
  - Register address map, [918](#)
- B0WUPTHRC\_ADDRESS
  - Register address map, [919](#)
- B1DRVS\_ADDRESS
  - Register address map, [919](#)
- B1FIE\_ADDRESS
  - Register address map, [919](#)
- B1FIS\_ADDRESS
  - Register address map, [919](#)
- B1OLDTC\_ADDRESS
  - Register address map, [919](#)
- B1SCDTC\_ADDRESS
  - Register address map, [919](#)
- B1WUPIE\_ADDRESS
  - Register address map, [919](#)
- B1WUPIS\_ADDRESS
  - Register address map, [919](#)
- B1WUPS\_ADDRESS
  - Register address map, [919](#)
- B1WUPTHRC\_ADDRESS
  - Register address map, [919](#)
- BDMMode
  - ftm\_user\_config\_t, [524](#)
- BITER
  - edma\_software\_tcd\_t, [382](#)
- BMETSC\_ADDRESS
  - Register address map, [919](#)
- BMHC\_ADDRESS
  - Register address map, [920](#)
- BMOIE\_DISABLE
  - Supply interrupt enable configuration options (0x1C), [976](#)
- BMOIE\_ENABLE
  - Supply interrupt enable configuration options (0x1C), [976](#)
- BMOTC\_ADDRESS
  - Register address map, [920](#)
- BMUIE\_DISABLE
  - Supply interrupt enable configuration options (0x1C), [976](#)
- BMUIE\_ENABLE
  - Supply interrupt enable configuration options (0x1C), [976](#)
- BMUTC\_ADDRESS
  - Register address map, [920](#)
- BSADC1\_ADDRESS
  - Register address map, [920](#)
- BSADC2\_ADDRESS
  - Register address map, [920](#)
- BUS\_ACTIVITY\_SET
  - Common Core API., [316](#)
- BUS\_CLK\_INDEX
  - Clock Manager Driver, [302](#)
- Backward Compatibility Symbols for S32K142, [243](#)
- Bank N fail interrupt enable configuration options (0x38/0x48), [244](#)
  - IO15OLIE\_DISABLE, [245](#)
  - IO15OLIE\_ENABLE, [245](#)
  - IO15SCIE\_DISABLE, [245](#)
  - IO15SCIE\_ENABLE, [245](#)
  - IO26OLIE\_DISABLE, [245](#)
  - IO26OLIE\_ENABLE, [245](#)
  - IO26SCIE\_DISABLE, [245](#)
  - IO26SCIE\_ENABLE, [246](#)
  - IO37OLIE\_DISABLE, [246](#)
  - IO37OLIE\_ENABLE, [246](#)
  - IO37SCIE\_DISABLE, [246](#)
  - IO37SCIE\_ENABLE, [246](#)
  - IO48OLIE\_DISABLE, [246](#)
  - IO48OLIE\_ENABLE, [246](#)
  - IO48SCIE\_DISABLE, [246](#)
  - IO48SCIE\_ENABLE, [246](#)
- Bank N wake-up interrupt enable configuration options (0x37/0x47), [247](#)
  - IO15FEIE\_DISABLE, [248](#)
  - IO15FEIE\_ENABLE, [248](#)
  - IO15REIE\_DISABLE, [248](#)
  - IO15REIE\_ENABLE, [248](#)
  - IO26FEIE\_DISABLE, [248](#)
  - IO26FEIE\_ENABLE, [248](#)
  - IO26REIE\_DISABLE, [248](#)
  - IO26REIE\_ENABLE, [249](#)
  - IO37FEIE\_DISABLE, [249](#)
  - IO37FEIE\_ENABLE, [249](#)
  - IO37REIE\_DISABLE, [249](#)
  - IO37REIE\_ENABLE, [249](#)
  - IO48FEIE\_DISABLE, [249](#)
  - IO48FEIE\_ENABLE, [249](#)
  - IO48REIE\_DISABLE, [249](#)
  - IO48REIE\_ENABLE, [249](#)
- bankFailIntMask
  - uja113x\_hvio\_bank\_cfg\_t, [143](#)
- bankWupIntMask
  - uja113x\_hvio\_bank\_cfg\_t, [143](#)
- bankWupThr
  - uja113x\_hvio\_bank\_cfg\_t, [143](#)
- baud\_rate
  - lin\_protocol\_state\_t, [769](#)
- baudRate
  - flexio\_i2c\_master\_user\_config\_t, [468](#)
  - flexio\_i2s\_master\_user\_config\_t, [478](#)
  - flexio\_spi\_master\_user\_config\_t, [495](#)
  - flexio\_uart\_user\_config\_t, [509](#)
  - i2c\_master\_t, [625](#)
  - i2s\_user\_config\_t, [604](#)
  - lin\_user\_config\_t, [653](#)
  - lpi2c\_baud\_rate\_params\_t, [676](#)
  - lpi2c\_master\_user\_config\_t, [674](#)
  - lpuart\_user\_config\_t, [735](#)
  - spi\_master\_t, [957](#)
  - uart\_user\_config\_t, [1052](#)
- baudrateEvalEnable

- lin\_state\_t, [655](#)
- bitCount
  - flexio\_uart\_user\_config\_t, [509](#)
  - uart\_user\_config\_t, [1052](#)
- bitCountPerChar
  - lpuart\_state\_t, [733](#)
  - lpuart\_user\_config\_t, [735](#)
- bitOrder
  - flexio\_spi\_master\_user\_config\_t, [495](#)
  - flexio\_spi\_slave\_user\_config\_t, [497](#)
  - spi\_master\_t, [957](#)
  - spi\_slave\_t, [958](#)
- bitcount
  - lpspi\_master\_config\_t, [703](#)
  - lpspi\_slave\_config\_t, [708](#)
- bitrate
  - flexcan\_user\_config\_t, [451](#)
- bitsPerFrame
  - lpspi\_state\_t, [705](#)
- bitsPerSec
  - lpspi\_master\_config\_t, [703](#)
- bitsWidth
  - flexio\_i2s\_master\_user\_config\_t, [478](#)
  - flexio\_i2s\_slave\_user\_config\_t, [479](#)
- bmOverVoltThr
  - uja113x\_bmc\_cfg\_t, [166](#)
- bmOverVoltThrRel
  - uja113x\_bmc\_cfg\_t, [166](#)
- bmSrcCtrl
  - uja113x\_bmc\_cfg\_t, [166](#)
- bmUndrVoltThr
  - uja113x\_bmc\_cfg\_t, [166](#)
- bmUndrVoltThrRel
  - uja113x\_bmc\_cfg\_t, [166](#)
- bmcBatMonitorCfg
  - uja113x\_init\_cfg\_t, [145](#)
- boldtBankOpenLoad
  - uja113x\_hvio\_bank\_cfg\_t, [143](#)
- brownOutCode
  - Flash Memory (Flash), [434](#)
- bscdtBankShortCirc
  - uja113x\_hvio\_bank\_cfg\_t, [143](#)
- bus\_activity
  - lin\_word\_status\_str\_t, [756](#)
- bypassPrescaler
  - extension\_lptmr\_for\_timer\_t, [998](#)
  - lptmr\_config\_t, [722](#)
- bytesPerFrame
  - lpspi\_state\_t, [705](#)
- CALLBACK\_HANDLER
  - Low level API, [771](#)
- CAN\_AbortTransfer
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [352](#)
- CAN\_CLK\_SOURCE\_OSC
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [351](#)
- CAN\_CLK\_SOURCE\_PERIPH
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [351](#)
- CAN\_ConfigRemoteResponseBuff
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [352](#)
- CAN\_ConfigRxBuff
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [353](#)
- CAN\_ConfigTxBuff
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [353](#)
- CAN\_DISABLE\_MODE
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [352](#)
- CAN\_Deinit
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [354](#)
- CAN\_FD\_DATA\_BITRATE
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [351](#)
- CAN\_GetBitrate
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [354](#)
- CAN\_GetDefaultConfig
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [354](#)
- CAN\_GetTransferStatus
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [354](#)
- CAN\_Init
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [355](#)
- CAN\_InstallEventCallback
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [355](#)
- CAN\_LOOPBACK\_MODE
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [352](#)
- CAN\_MSG\_ID\_EXT
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [352](#)
- CAN\_MSG\_ID\_STD
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [352](#)
- CAN\_NOMINAL\_BITRATE
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [351](#)
- CAN\_NORMAL\_MODE
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [352](#)
- CAN\_PAYLOAD\_SIZE\_16
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [351](#)
- CAN\_PAYLOAD\_SIZE\_32
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [352](#)
- CAN\_PAYLOAD\_SIZE\_64
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [352](#)

- Layer (CAN PAL), [352](#)
- CAN\_PAYLOAD\_SIZE\_8
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [351](#)
- CAN\_Receive
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [355](#)
- CAN\_ReceiveBlocking
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [356](#)
- CAN\_Send
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [356](#)
- CAN\_SendBlocking
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [357](#)
- CAN\_SetBtrRate
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [357](#)
- CAN\_SetRxFilt
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [357](#)
- CANC\_ADDRESS
  - Register address map, [920](#)
- CBSIE\_DISABLE
  - Transceiver interrupt enable configuration options (0x23), [1004](#)
- CBSIE\_ENABLE
  - Transceiver interrupt enable configuration options (0x23), [1004](#)
- CDR\_DATA\_RATE\_100
  - 01. CAN & CAN partial networking configuration, [156](#)
- CDR\_DATA\_RATE\_1000
  - 01. CAN & CAN partial networking configuration, [156](#)
- CDR\_DATA\_RATE\_125
  - 01. CAN & CAN partial networking configuration, [156](#)
- CDR\_DATA\_RATE\_250
  - 01. CAN & CAN partial networking configuration, [156](#)
- CDR\_DATA\_RATE\_50
  - 01. CAN & CAN partial networking configuration, [156](#)
- CDR\_DATA\_RATE\_500a
  - 01. CAN & CAN partial networking configuration, [156](#)
- CDR\_DATA\_RATE\_500b
  - 01. CAN & CAN partial networking configuration, [156](#)
- CDR\_DATA\_RATE\_500c
  - 01. CAN & CAN partial networking configuration, [156](#)
- CFIE\_DISABLE
  - Transceiver interrupt enable configuration options (0x23), [1004](#)
- CFIE\_ENABLE
  - Transceiver interrupt enable configuration options (0x23), [1004](#)
- CHAN0\_IDX
  - FlexTimer (FTM), [525](#)
- CHAN1\_IDX
  - FlexTimer (FTM), [525](#)
- CHAN2\_IDX
  - FlexTimer (FTM), [525](#)
- CHAN3\_IDX
  - FlexTimer (FTM), [525](#)
- CHAN4\_IDX
  - FlexTimer (FTM), [525](#)
- CHAN5\_IDX
  - FlexTimer (FTM), [525](#)
- CHAN6\_IDX
  - FlexTimer (FTM), [525](#)
- CHAN7\_IDX
  - FlexTimer (FTM), [525](#)
- CHECK\_PARITY
  - LIN Driver, [657](#)
- CITER
  - edma\_software\_tcd\_t, [382](#)
- CLEAR\_FTFx\_FSTAT\_ERROR\_BITS
  - Flash Memory (Flash), [421](#)
- CLK\_SRC\_FIRC
  - Clock Manager Driver, [302](#)
- CLK\_SRC\_FIRC\_DIV1
  - Clock Manager Driver, [302](#)
- CLK\_SRC\_FIRC\_DIV2
  - Clock Manager Driver, [302](#)
- CLK\_SRC\_OFF
  - Clock Manager Driver, [302](#)
- CLK\_SRC\_SIRC
  - Clock Manager Driver, [302](#)
- CLK\_SRC\_SIRC\_DIV1
  - Clock Manager Driver, [302](#)
- CLK\_SRC\_SIRC\_DIV2
  - Clock Manager Driver, [302](#)
- CLK\_SRC\_SOSC
  - Clock Manager Driver, [302](#)
- CLK\_SRC\_SOSC\_DIV1
  - Clock Manager Driver, [302](#)
- CLK\_SRC\_SOSC\_DIV2
  - Clock Manager Driver, [302](#)
- CLK\_SRC\_SPLL
  - Clock Manager Driver, [303](#)
- CLK\_SRC\_SPLL\_DIV1
  - Clock Manager Driver, [303](#)
- CLK\_SRC\_SPLL\_DIV2
  - Clock Manager Driver, [303](#)
- CLOCK\_DRV\_GetFreq
  - Clock, [276](#)
- CLOCK\_DRV\_GetSystemClockSource
  - Clock Manager Driver, [310](#)
- CLOCK\_DRV\_Init
  - Clock, [276](#)
- CLOCK\_DRV\_SetClockSource
  - Clock Manager Driver, [310](#)

- CLOCK\_DRV\_SetModuleClock
  - Clock Manager Driver, [310](#)
- CLOCK\_DRV\_SetSystemClock
  - Clock Manager Driver, [312](#)
- CLOCK\_MANAGER\_CALLBACK\_AFTER
  - Clock Manager Driver, [304](#)
- CLOCK\_MANAGER\_CALLBACK\_BEFORE
  - Clock Manager Driver, [304](#)
- CLOCK\_MANAGER\_CALLBACK\_BEFORE\_AFTER
  - Clock Manager Driver, [304](#)
- CLOCK\_MANAGER\_NOTIFY\_AFTER
  - Clock Manager Driver, [304](#)
- CLOCK\_MANAGER\_NOTIFY\_BEFORE
  - Clock Manager Driver, [304](#)
- CLOCK\_MANAGER\_NOTIFY\_RECOVER
  - Clock Manager Driver, [304](#)
- CLOCK\_MANAGER\_POLICY\_AGREEMENT
  - Clock Manager Driver, [304](#)
- CLOCK\_MANAGER\_POLICY\_FORCIBLE
  - Clock Manager Driver, [304](#)
- CLOCK\_SYS\_GetCurrentConfiguration
  - Clock Manager Driver, [312](#)
- CLOCK\_SYS\_GetErrorCallback
  - Clock Manager Driver, [312](#)
- CLOCK\_SYS\_GetFreq
  - Clock Manager Driver, [312](#)
- CLOCK\_SYS\_Init
  - Clock Manager Driver, [312](#)
- CLOCK\_SYS\_SetConfiguration
  - Clock Manager Driver, [314](#)
- CLOCK\_SYS\_UpdateConfiguration
  - Clock Manager Driver, [314](#)
- CLOCK\_TRACE\_SRC\_CORE\_CLK
  - Clock Manager Driver, [304](#)
- CMC\_TRANSCEIVER\_ACTIVE
  - 01. CAN & CAN partial networking configuration, [155](#)
- CMC\_TRANSCEIVER\_ACTIVE2
  - 01. CAN & CAN partial networking configuration, [155](#)
- CMC\_TRANSCEIVER\_LISTEN
  - 01. CAN & CAN partial networking configuration, [155](#)
- CMC\_TRANSCEIVER\_OFF
  - 01. CAN & CAN partial networking configuration, [155](#)
- CMP\_AVAILABLE
  - Comparator Driver, [333](#)
- CMP\_BOTH\_EDGES
  - Comparator Driver, [334](#)
- CMP\_CONTINUOUS
  - Comparator Driver, [333](#)
- CMP\_COUT
  - Comparator Driver, [334](#)
- CMP\_COUTA
  - Comparator Driver, [334](#)
- CMP\_DAC
  - Comparator Driver, [334](#)
- CMP\_DISABLED
  - Comparator Driver, [333](#)
- CMP\_DRV\_ClearInputFlags
  - Comparator Driver, [335](#)
- CMP\_DRV\_ClearOutputFlags
  - Comparator Driver, [335](#)
- CMP\_DRV\_ConfigComparator
  - Comparator Driver, [335](#)
- CMP\_DRV\_ConfigDAC
  - Comparator Driver, [335](#)
- CMP\_DRV\_ConfigMUX
  - Comparator Driver, [336](#)
- CMP\_DRV\_ConfigTriggerMode
  - Comparator Driver, [336](#)
- CMP\_DRV\_GetComparatorConfig
  - Comparator Driver, [336](#)
- CMP\_DRV\_GetConfigAll
  - Comparator Driver, [337](#)
- CMP\_DRV\_GetDACConfig
  - Comparator Driver, [337](#)
- CMP\_DRV\_GetDefaultConfig
  - Comparator Driver, [337](#)
- CMP\_DRV\_GetInitConfigAll
  - Comparator Driver, [337](#)
- CMP\_DRV\_GetInitConfigComparator
  - Comparator Driver, [338](#)
- CMP\_DRV\_GetInitConfigDAC
  - Comparator Driver, [338](#)
- CMP\_DRV\_GetInitConfigMUX
  - Comparator Driver, [338](#)
- CMP\_DRV\_GetInitTriggerMode
  - Comparator Driver, [339](#)
- CMP\_DRV\_GetInputFlags
  - Comparator Driver, [339](#)
- CMP\_DRV\_GetMUXConfig
  - Comparator Driver, [339](#)
- CMP\_DRV\_GetOutputFlags
  - Comparator Driver, [339](#)
- CMP\_DRV\_GetTriggerModeConfig
  - Comparator Driver, [340](#)
- CMP\_DRV\_Init
  - Comparator Driver, [340](#)
- CMP\_DRV\_Reset
  - Comparator Driver, [340](#)
- CMP\_FALLING\_EDGE
  - Comparator Driver, [334](#)
- CMP\_HIGH\_SPEED
  - Comparator Driver, [334](#)
- CMP\_INPUT\_FLAGS\_MASK
  - Comparator Driver, [332](#)
- CMP\_INPUT\_FLAGS\_SHIFT
  - Comparator Driver, [332](#)
- CMP\_INVERT
  - Comparator Driver, [333](#)
- CMP\_LEVEL\_HYS\_0
  - Comparator Driver, [333](#)
- CMP\_LEVEL\_HYS\_1
  - Comparator Driver, [333](#)

- CMP\_LEVEL\_HYS\_2
  - Comparator Driver, [333](#)
- CMP\_LEVEL\_HYS\_3
  - Comparator Driver, [333](#)
- CMP\_LOW\_SPEED
  - Comparator Driver, [334](#)
- CMP\_MINUS\_FIXED
  - Comparator Driver, [332](#)
- CMP\_MUX
  - Comparator Driver, [334](#)
- CMP\_NO\_EVENT
  - Comparator Driver, [334](#)
- CMP\_NORMAL
  - Comparator Driver, [333](#)
- CMP\_PLUS\_FIXED
  - Comparator Driver, [332](#)
- CMP\_RISING\_EDGE
  - Comparator Driver, [334](#)
- CMP\_ROUND\_ROBIN\_CHANNELS\_MASK
  - Comparator Driver, [332](#)
- CMP\_ROUND\_ROBIN\_CHANNELS\_SHIFT
  - Comparator Driver, [332](#)
- CMP\_SAMPLED\_FILTERED\_EXT\_CLK
  - Comparator Driver, [333](#)
- CMP\_SAMPLED\_FILTERED\_INT\_CLK
  - Comparator Driver, [333](#)
- CMP\_SAMPLED\_NONFILTERED\_EXT\_CLK
  - Comparator Driver, [333](#)
- CMP\_SAMPLED\_NONFILTERED\_INT\_CLK
  - Comparator Driver, [333](#)
- CMP\_UNAVAILABLE
  - Comparator Driver, [333](#)
- CMP\_VIN1
  - Comparator Driver, [334](#)
- CMP\_VIN2
  - Comparator Driver, [334](#)
- CMP\_WINDOWED
  - Comparator Driver, [333](#)
- CMP\_WINDOWED\_FILTERED
  - Comparator Driver, [333](#)
- CMP\_WINDOWED\_RESAMPLED
  - Comparator Driver, [333](#)
- CORE\_CLK\_INDEX
  - Clock Manager Driver, [303](#)
- CRC Driver, [250](#)
  - CRC\_DRV\_Configure, [251](#)
  - CRC\_DRV\_Deinit, [251](#)
  - CRC\_DRV\_GetConfig, [252](#)
  - CRC\_DRV\_GetCrc16, [252](#)
  - CRC\_DRV\_GetCrc32, [252](#)
  - CRC\_DRV\_GetCrc8, [253](#)
  - CRC\_DRV\_GetCrcResult, [253](#)
  - CRC\_DRV\_GetDefaultConfig, [253](#)
  - CRC\_DRV\_Init, [254](#)
  - CRC\_DRV\_WriteData, [254](#)
  - CRC\_TRANSPOSE\_BITS, [251](#)
  - CRC\_TRANSPOSE\_BITS\_AND\_BYTES, [251](#)
  - CRC\_TRANSPOSE\_BYTES, [251](#)
- CRC\_TRANSPOSE\_NONE, [251](#)
  - crc\_transpose\_t, [251](#)
- CRC\_DRV\_Configure
  - CRC Driver, [251](#)
- CRC\_DRV\_Deinit
  - CRC Driver, [251](#)
- CRC\_DRV\_GetConfig
  - CRC Driver, [252](#)
- CRC\_DRV\_GetCrc16
  - CRC Driver, [252](#)
- CRC\_DRV\_GetCrc32
  - CRC Driver, [252](#)
- CRC\_DRV\_GetCrc8
  - CRC Driver, [253](#)
- CRC\_DRV\_GetCrcResult
  - CRC Driver, [253](#)
- CRC\_DRV\_GetDefaultConfig
  - CRC Driver, [253](#)
- CRC\_DRV\_Init
  - CRC Driver, [254](#)
- CRC\_DRV\_WriteData
  - CRC Driver, [254](#)
- CRC\_TRANSPOSE\_BITS
  - CRC Driver, [251](#)
- CRC\_TRANSPOSE\_BITS\_AND\_BYTES
  - CRC Driver, [251](#)
- CRC\_TRANSPOSE\_BYTES
  - CRC Driver, [251](#)
- CRC\_TRANSPOSE\_NONE
  - CRC Driver, [251](#)
- CSC\_NONE
  - 01. CAN & CAN partial networking configuration, [155](#)
- CSC\_OVERVOLTAGE
  - 01. CAN & CAN partial networking configuration, [155](#)
- CSC\_UNDRV\_OVERV
  - 01. CAN & CAN partial networking configuration, [155](#)
- CSC\_UNDRVOLTAGE
  - 01. CAN & CAN partial networking configuration, [155](#)
- CSE\_KEY\_SIZE\_CODE\_MAX
  - Flash Memory (Flash), [421](#)
- CSEC\_BOOT\_MAC
  - CSEc Driver, [264](#)
- CSEC\_BOOT\_MAC\_KEY
  - CSEc Driver, [264](#)
- CSEC\_BOOT\_NOT\_DEFINED
  - CSEc Driver, [263](#)
- CSEC\_BOOT\_PARALLEL
  - CSEc Driver, [263](#)
- CSEC\_BOOT\_SERIAL
  - CSEc Driver, [263](#)
- CSEC\_BOOT\_STRICT
  - CSEc Driver, [263](#)
- CSEC\_CALL\_SEQ\_FIRST
  - CSEc Driver, [264](#)

- CSEC\_CALL\_SEQ\_SUBSEQUENT
  - CSEc Driver, [264](#)
- CSEC\_CMD\_BOOT\_DEFINE
  - CSEc Driver, [264](#)
- CSEC\_CMD\_BOOT\_FAILURE
  - CSEc Driver, [264](#)
- CSEC\_CMD\_BOOT\_OK
  - CSEc Driver, [264](#)
- CSEC\_CMD\_DBG\_AUTH
  - CSEc Driver, [264](#)
- CSEC\_CMD\_DBG\_CHAL
  - CSEc Driver, [264](#)
- CSEC\_CMD\_DEC\_CBC
  - CSEc Driver, [264](#)
- CSEC\_CMD\_DEC\_ECB
  - CSEc Driver, [264](#)
- CSEC\_CMD\_ENC\_CBC
  - CSEc Driver, [264](#)
- CSEC\_CMD\_ENC\_ECB
  - CSEc Driver, [264](#)
- CSEC\_CMD\_EXPORT\_RAM\_KEY
  - CSEc Driver, [264](#)
- CSEC\_CMD\_EXTEND\_SEED
  - CSEc Driver, [264](#)
- CSEC\_CMD\_GENERATE\_MAC
  - CSEc Driver, [264](#)
- CSEC\_CMD\_GET\_ID
  - CSEc Driver, [264](#)
- CSEC\_CMD\_INIT\_RNG
  - CSEc Driver, [264](#)
- CSEC\_CMD\_LOAD\_KEY
  - CSEc Driver, [264](#)
- CSEC\_CMD\_LOAD\_PLAIN\_KEY
  - CSEc Driver, [264](#)
- CSEC\_CMD\_MP\_COMPRESS
  - CSEc Driver, [264](#)
- CSEC\_CMD\_RESERVED\_1
  - CSEc Driver, [264](#)
- CSEC\_CMD\_RESERVED\_2
  - CSEc Driver, [264](#)
- CSEC\_CMD\_RESERVED\_3
  - CSEc Driver, [264](#)
- CSEC\_CMD\_RND
  - CSEc Driver, [264](#)
- CSEC\_CMD\_VERIFY\_MAC
  - CSEc Driver, [264](#)
- CSEC\_DRV\_BootDefine
  - CSEc Driver, [265](#)
- CSEC\_DRV\_BootFailure
  - CSEc Driver, [265](#)
- CSEC\_DRV\_BootOK
  - CSEc Driver, [265](#)
- CSEC\_DRV\_CancelCommand
  - CSEc Driver, [266](#)
- CSEC\_DRV\_DbgAuth
  - CSEc Driver, [266](#)
- CSEC\_DRV\_DbgChal
  - CSEc Driver, [266](#)
- CSEC\_DRV\_DecryptCBC
  - CSEc Driver, [266](#)
- CSEC\_DRV\_DecryptCBCAsync
  - CSEc Driver, [267](#)
- CSEC\_DRV\_DecryptECB
  - CSEc Driver, [267](#)
- CSEC\_DRV\_DecryptECBAsync
  - CSEc Driver, [268](#)
- CSEC\_DRV\_Deinit
  - CSEc Driver, [268](#)
- CSEC\_DRV\_EncryptCBC
  - CSEc Driver, [268](#)
- CSEC\_DRV\_EncryptCBCAsync
  - CSEc Driver, [268](#)
- CSEC\_DRV\_EncryptECB
  - CSEc Driver, [269](#)
- CSEC\_DRV\_EncryptECBAsync
  - CSEc Driver, [269](#)
- CSEC\_DRV\_ExportRAMKey
  - CSEc Driver, [270](#)
- CSEC\_DRV\_ExtendSeed
  - CSEc Driver, [270](#)
- CSEC\_DRV\_GenerateMAC
  - CSEc Driver, [270](#)
- CSEC\_DRV\_GenerateMACAddrMode
  - CSEc Driver, [270](#)
- CSEC\_DRV\_GenerateMACAsync
  - CSEc Driver, [271](#)
- CSEC\_DRV\_GenerateRND
  - CSEc Driver, [271](#)
- CSEC\_DRV\_GetAsyncCmdStatus
  - CSEc Driver, [271](#)
- CSEC\_DRV\_GetID
  - CSEc Driver, [272](#)
- CSEC\_DRV\_GetStatus
  - CSEc Driver, [272](#)
- CSEC\_DRV\_Init
  - CSEc Driver, [272](#)
- CSEC\_DRV\_InitRNG
  - CSEc Driver, [272](#)
- CSEC\_DRV\_InstallCallback
  - CSEc Driver, [272](#)
- CSEC\_DRV\_LoadKey
  - CSEc Driver, [273](#)
- CSEC\_DRV\_LoadPlainKey
  - CSEc Driver, [273](#)
- CSEC\_DRV\_MPCompress
  - CSEc Driver, [273](#)
- CSEC\_DRV\_VerifyMAC
  - CSEc Driver, [274](#)
- CSEC\_DRV\_VerifyMACAddrMode
  - CSEc Driver, [274](#)
- CSEC\_DRV\_VerifyMACAsync
  - CSEc Driver, [275](#)
- CSEC\_KEY\_1
  - CSEc Driver, [264](#)
- CSEC\_KEY\_10
  - CSEc Driver, [265](#)



CSEC\_KEY\_11  
    CSEc Driver, [265](#)  
CSEC\_KEY\_12  
    CSEc Driver, [265](#)  
CSEC\_KEY\_13  
    CSEc Driver, [265](#)  
CSEC\_KEY\_14  
    CSEc Driver, [265](#)  
CSEC\_KEY\_15  
    CSEc Driver, [265](#)  
CSEC\_KEY\_16  
    CSEc Driver, [265](#)  
CSEC\_KEY\_17  
    CSEc Driver, [265](#)  
CSEC\_KEY\_2  
    CSEc Driver, [264](#)  
CSEC\_KEY\_3  
    CSEc Driver, [265](#)  
CSEC\_KEY\_4  
    CSEc Driver, [265](#)  
CSEC\_KEY\_5  
    CSEc Driver, [265](#)  
CSEC\_KEY\_6  
    CSEc Driver, [265](#)  
CSEC\_KEY\_7  
    CSEc Driver, [265](#)  
CSEC\_KEY\_8  
    CSEc Driver, [265](#)  
CSEC\_KEY\_9  
    CSEc Driver, [265](#)  
CSEC\_MASTER\_ECU  
    CSEc Driver, [264](#)  
CSEC\_RAM\_KEY  
    CSEc Driver, [265](#)  
CSEC\_SECRET\_KEY  
    CSEc Driver, [264](#)  
CSEC\_STATUS\_BOOT\_FINISHED  
    CSEc Driver, [262](#)  
CSEC\_STATUS\_BOOT\_INIT  
    CSEc Driver, [262](#)  
CSEC\_STATUS\_BOOT\_OK  
    CSEc Driver, [262](#)  
CSEC\_STATUS\_BUSY  
    CSEc Driver, [262](#)  
CSEC\_STATUS\_EXT\_DEBUGGER  
    CSEc Driver, [263](#)  
CSEC\_STATUS\_INT\_DEBUGGER  
    CSEc Driver, [263](#)  
CSEC\_STATUS\_RND\_INIT  
    CSEc Driver, [263](#)  
CSEC\_STATUS\_SECURE\_BOOT  
    CSEc Driver, [263](#)  
CSEc Driver, [255](#)  
    CSEC\_BOOT\_MAC, [264](#)  
    CSEC\_BOOT\_MAC\_KEY, [264](#)  
    CSEC\_BOOT\_NOT\_DEFINED, [263](#)  
    CSEC\_BOOT\_PARALLEL, [263](#)  
    CSEC\_BOOT\_SERIAL, [263](#)  
    CSEC\_BOOT\_STRICT, [263](#)  
    CSEC\_CALL\_SEQ\_FIRST, [264](#)  
    CSEC\_CALL\_SEQ\_SUBSEQUENT, [264](#)  
    CSEC\_CMD\_BOOT\_DEFINE, [264](#)  
    CSEC\_CMD\_BOOT\_FAILURE, [264](#)  
    CSEC\_CMD\_BOOT\_OK, [264](#)  
    CSEC\_CMD\_DBG\_AUTH, [264](#)  
    CSEC\_CMD\_DBG\_CHAL, [264](#)  
    CSEC\_CMD\_DEC\_CBC, [264](#)  
    CSEC\_CMD\_DEC\_ECB, [264](#)  
    CSEC\_CMD\_ENC\_CBC, [264](#)  
    CSEC\_CMD\_ENC\_ECB, [264](#)  
    CSEC\_CMD\_EXPORT\_RAM\_KEY, [264](#)  
    CSEC\_CMD\_EXTEND\_SEED, [264](#)  
    CSEC\_CMD\_GENERATE\_MAC, [264](#)  
    CSEC\_CMD\_GET\_ID, [264](#)  
    CSEC\_CMD\_INIT\_RNG, [264](#)  
    CSEC\_CMD\_LOAD\_KEY, [264](#)  
    CSEC\_CMD\_LOAD\_PLAIN\_KEY, [264](#)  
    CSEC\_CMD\_MP\_COMPRESS, [264](#)  
    CSEC\_CMD\_RESERVED\_1, [264](#)  
    CSEC\_CMD\_RESERVED\_2, [264](#)  
    CSEC\_CMD\_RESERVED\_3, [264](#)  
    CSEC\_CMD\_RND, [264](#)  
    CSEC\_CMD\_VERIFY\_MAC, [264](#)  
    CSEC\_DRV\_BootDefine, [265](#)  
    CSEC\_DRV\_BootFailure, [265](#)  
    CSEC\_DRV\_BootOK, [265](#)  
    CSEC\_DRV\_CancelCommand, [266](#)  
    CSEC\_DRV\_DbgAuth, [266](#)  
    CSEC\_DRV\_DbgChal, [266](#)  
    CSEC\_DRV\_DecryptCBC, [266](#)  
    CSEC\_DRV\_DecryptCBCAsync, [267](#)  
    CSEC\_DRV\_DecryptECB, [267](#)  
    CSEC\_DRV\_DecryptECBAsync, [268](#)  
    CSEC\_DRV\_Deinit, [268](#)  
    CSEC\_DRV\_EncryptCBC, [268](#)  
    CSEC\_DRV\_EncryptCBCAsync, [268](#)  
    CSEC\_DRV\_EncryptECB, [269](#)  
    CSEC\_DRV\_EncryptECBAsync, [269](#)  
    CSEC\_DRV\_ExportRAMKey, [270](#)  
    CSEC\_DRV\_ExtendSeed, [270](#)  
    CSEC\_DRV\_GenerateMAC, [270](#)  
    CSEC\_DRV\_GenerateMACAddrMode, [270](#)  
    CSEC\_DRV\_GenerateMACAsync, [271](#)  
    CSEC\_DRV\_GenerateRND, [271](#)  
    CSEC\_DRV\_GetAsyncCmdStatus, [271](#)  
    CSEC\_DRV\_GetID, [272](#)  
    CSEC\_DRV\_GetStatus, [272](#)  
    CSEC\_DRV\_Init, [272](#)  
    CSEC\_DRV\_InitRNG, [272](#)  
    CSEC\_DRV\_InstallCallback, [272](#)  
    CSEC\_DRV\_LoadKey, [273](#)  
    CSEC\_DRV\_LoadPlainKey, [273](#)  
    CSEC\_DRV\_MPCompress, [273](#)  
    CSEC\_DRV\_VerifyMAC, [274](#)  
    CSEC\_DRV\_VerifyMACAddrMode, [274](#)  
    CSEC\_DRV\_VerifyMACAsync, [275](#)

- CSEC\_KEY\_1, [264](#)
- CSEC\_KEY\_10, [265](#)
- CSEC\_KEY\_11, [265](#)
- CSEC\_KEY\_12, [265](#)
- CSEC\_KEY\_13, [265](#)
- CSEC\_KEY\_14, [265](#)
- CSEC\_KEY\_15, [265](#)
- CSEC\_KEY\_16, [265](#)
- CSEC\_KEY\_17, [265](#)
- CSEC\_KEY\_2, [264](#)
- CSEC\_KEY\_3, [265](#)
- CSEC\_KEY\_4, [265](#)
- CSEC\_KEY\_5, [265](#)
- CSEC\_KEY\_6, [265](#)
- CSEC\_KEY\_7, [265](#)
- CSEC\_KEY\_8, [265](#)
- CSEC\_KEY\_9, [265](#)
- CSEC\_MASTER\_ECU, [264](#)
- CSEC\_RAM\_KEY, [265](#)
- CSEC\_SECRET\_KEY, [264](#)
- CSEC\_STATUS\_BOOT\_FINISHED, [262](#)
- CSEC\_STATUS\_BOOT\_INIT, [262](#)
- CSEC\_STATUS\_BOOT\_OK, [262](#)
- CSEC\_STATUS\_BUSY, [262](#)
- CSEC\_STATUS\_EXT\_DEBUGGER, [263](#)
- CSEC\_STATUS\_INT\_DEBUGGER, [263](#)
- CSEC\_STATUS\_RND\_INIT, [263](#)
- CSEC\_STATUS\_SECURE\_BOOT, [263](#)
- csec\_boot\_flavor\_t, [263](#)
- csec\_call\_sequence\_t, [263](#)
- csec\_cmd\_t, [264](#)
- csec\_key\_id\_t, [264](#)
- csec\_status\_t, [263](#)
- CSR
  - edma\_software\_tcd\_t, [382](#)
- CWIE\_DISABLE
  - Transceiver interrupt enable configuration options (0x23), [1004](#)
- CWIE\_ENABLE
  - Transceiver interrupt enable configuration options (0x23), [1004](#)
- CallBack
  - Flash Memory (Flash), [434](#)
- Callback
  - lin\_state\_t, [655](#)
- callback
  - adc\_group\_config\_t, [234](#)
  - clock\_manager\_callback\_user\_config\_t, [301](#)
  - csec\_state\_t, [261](#)
  - edma\_channel\_config\_t, [378](#)
  - edma\_chn\_state\_t, [377](#)
  - FlexCANState, [448](#)
  - flexio\_i2c\_master\_user\_config\_t, [468](#)
  - flexio\_i2s\_master\_user\_config\_t, [478](#)
  - flexio\_i2s\_slave\_user\_config\_t, [479](#)
  - flexio\_spi\_master\_user\_config\_t, [495](#)
  - flexio\_spi\_slave\_user\_config\_t, [497](#)
  - flexio\_uart\_user\_config\_t, [509](#)
  - i2c\_master\_t, [625](#)
  - i2c\_slave\_t, [626](#)
  - i2s\_user\_config\_t, [604](#)
  - lpspi\_master\_config\_t, [704](#)
  - lpspi\_slave\_config\_t, [708](#)
  - lpspi\_state\_t, [705](#)
  - security\_user\_config\_t, [935](#)
  - spi\_master\_t, [957](#)
  - spi\_slave\_t, [959](#)
  - timer\_chan\_config\_t, [997](#)
- callbackConfig
  - clock\_manager\_state\_t, [301](#)
- callbackData
  - clock\_manager\_callback\_user\_config\_t, [301](#)
  - power\_manager\_callback\_user\_config\_t, [867](#)
- callbackFunction
  - power\_manager\_callback\_user\_config\_t, [867](#)
- callbackNum
  - clock\_manager\_state\_t, [301](#)
- callbackParam
  - csec\_state\_t, [261](#)
  - edma\_channel\_config\_t, [378](#)
  - FlexCANState, [448](#)
  - flexio\_i2c\_master\_user\_config\_t, [468](#)
  - flexio\_i2s\_master\_user\_config\_t, [478](#)
  - flexio\_i2s\_slave\_user\_config\_t, [479](#)
  - flexio\_spi\_master\_user\_config\_t, [495](#)
  - flexio\_spi\_slave\_user\_config\_t, [497](#)
  - flexio\_uart\_user\_config\_t, [509](#)
  - i2c\_master\_t, [625](#)
  - i2c\_slave\_t, [626](#)
  - i2s\_user\_config\_t, [605](#)
  - lpi2c\_master\_user\_config\_t, [674](#)
  - lpi2c\_slave\_user\_config\_t, [675](#)
  - lpspi\_master\_config\_t, [704](#)
  - lpspi\_slave\_config\_t, [708](#)
  - lpspi\_state\_t, [706](#)
  - security\_user\_config\_t, [935](#)
  - spi\_master\_t, [957](#)
  - spi\_slave\_t, [959](#)
  - timer\_chan\_config\_t, [997](#)
- callbackParams
  - rtc\_alarm\_config\_t, [897](#)
  - rtc\_interrupt\_config\_t, [897](#)
- callbackType
  - clock\_manager\_callback\_user\_config\_t, [301](#)
  - power\_manager\_callback\_user\_config\_t, [867](#)
- callbackUserData
  - adc\_group\_config\_t, [234](#)
- can
  - sbc\_int\_config\_t, [1020](#)
- can\_bitrate\_phase\_t
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [351](#)
- can\_buff\_config\_t, [348](#)
- enableBRS, [348](#)
- enableFD, [348](#)
- fdPadding, [349](#)



- idType, [349](#)
- isRemote, [349](#)
- can\_clk\_source\_t
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [351](#)
- can\_fd\_payload\_size\_t
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [351](#)
- can\_instance\_t, [1082](#)
  - instIdx, [1082](#)
  - instType, [1082](#)
- can\_message\_t, [349](#)
  - cs, [349](#)
  - data, [349](#)
  - id, [349](#)
  - length, [349](#)
- can\_msg\_id\_type\_t
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [352](#)
- can\_operation\_modes\_t
  - Controller Area Network - Peripheral Abstraction Layer (CAN PAL), [352](#)
- can\_time\_segment\_t, [347](#)
  - phaseSeg1, [348](#)
  - phaseSeg2, [348](#)
  - preDivider, [348](#)
  - propSeg, [348](#)
  - rJumpwidth, [348](#)
- can\_user\_config\_t, [349](#)
  - dataBitrate, [350](#)
  - enableFD, [350](#)
  - extension, [350](#)
  - maxBuffNum, [350](#)
  - mode, [350](#)
  - nominalBitrate, [350](#)
  - payloadSize, [350](#)
  - peClkSrc, [350](#)
- canConf
  - sbc\_can\_conf\_t, [1018](#)
- canDataRate
  - uja113x\_cpnc\_cfg\_t, [152](#)
- canFDCtrl
  - uja113x\_canc\_cfg\_t, [151](#)
- canModeCtrl
  - uja113x\_canc\_cfg\_t, [151](#)
- canPartNetCtrl
  - uja113x\_canc\_cfg\_t, [151](#)
- canShutdownCtrl
  - uja113x\_canc\_cfg\_t, [152](#)
- canTransEvt
  - sbc\_can\_conf\_t, [1018](#)
- cancCanCfg
  - uja113x\_init\_cfg\_t, [145](#)
- cbs
  - sbc\_trans\_evnt\_stat\_t, [1026](#)
- cbse
  - sbc\_trans\_evnt\_t, [1017](#)
- cbss
  - sbc\_trans\_stat\_t, [1023](#)
- cf
  - sbc\_trans\_evnt\_stat\_t, [1026](#)
- cfdc
  - sbc\_can\_ctr\_t, [1016](#)
- cfe
  - sbc\_trans\_evnt\_t, [1017](#)
- cfs
  - sbc\_trans\_stat\_t, [1023](#)
- chMode
  - ftm\_output\_cmp\_ch\_param\_t, [567](#)
  - oc\_output\_ch\_param\_t, [838](#)
- chainChannel
  - lpit\_user\_channel\_config\_t, [690](#)
- chanConfigArray
  - timer\_config\_t, [997](#)
- chanType
  - timer\_chan\_config\_t, [997](#)
- channel
  - adc\_chan\_config\_t, [219](#)
  - eim\_user\_channel\_config\_t, [398](#)
  - erm\_user\_config\_t, [403](#)
  - pwm\_channel\_t, [887](#)
  - timer\_chan\_config\_t, [997](#)
- channel\_extension\_ftm\_for\_ic\_t, [616](#)
  - continuousModeEn, [616](#)
- channelCallbackParams
  - ic\_input\_ch\_param\_t, [615](#)
  - oc\_output\_ch\_param\_t, [838](#)
- channelCallbacks
  - ic\_input\_ch\_param\_t, [615](#)
  - oc\_output\_ch\_param\_t, [838](#)
- channelExtension
  - ic\_input\_ch\_param\_t, [615](#)
  - oc\_output\_ch\_param\_t, [838](#)
- channelPriority
  - edma\_channel\_config\_t, [378](#)
- channelType
  - pwm\_channel\_t, [887](#)
- channelsCallbacks
  - ftm\_input\_ch\_param\_t, [555](#)
  - ftm\_state\_t, [522](#)
- channelsCallbacksParams
  - ftm\_input\_ch\_param\_t, [555](#)
  - ftm\_state\_t, [522](#)
- check\_timeout
  - lin\_tl\_descriptor\_t, [763](#)
- check\_timeout\_type
  - lin\_tl\_descriptor\_t, [763](#)
- checkBitMask
  - eim\_user\_channel\_config\_t, [398](#)
- checksum
  - lin\_state\_t, [655](#)
- chipType
  - uja113x\_driver\_t, [147](#)
  - uja113x\_drv\_cfg\_t, [142](#)
- chnArbitration
  - edma\_user\_config\_t, [377](#)

- classicPID
  - lin\_user\_config\_t, 653
- clkGate
  - peripheral\_clock\_config\_t, 296
- clkPhase
  - lpspi\_master\_config\_t, 704
  - lpspi\_slave\_config\_t, 708
- clkPolarity
  - lpspi\_master\_config\_t, 704
  - lpspi\_slave\_config\_t, 708
- clkPreDiv
  - pdb\_timer\_config\_t, 849
- clkPreMultFactor
  - pdb\_timer\_config\_t, 849
- clkSource
  - wdg\_config\_t, 1063
  - wdog\_user\_config\_t, 1071
- clkSrc
  - peripheral\_clock\_config\_t, 296
- Clock, 276
  - CLOCK\_DRV\_GetFreq, 276
  - CLOCK\_DRV\_Init, 276
- Clock Manager, 277
- Clock Manager Driver, 278
  - ALL\_MODES, 305
  - BUS\_CLK\_INDEX, 302
  - CLK\_SRC\_FIRC, 302
  - CLK\_SRC\_FIRC\_DIV1, 302
  - CLK\_SRC\_FIRC\_DIV2, 302
  - CLK\_SRC\_OFF, 302
  - CLK\_SRC\_SIRC, 302
  - CLK\_SRC\_SIRC\_DIV1, 302
  - CLK\_SRC\_SIRC\_DIV2, 302
  - CLK\_SRC\_SOSC, 302
  - CLK\_SRC\_SOSC\_DIV1, 302
  - CLK\_SRC\_SOSC\_DIV2, 302
  - CLK\_SRC\_SPLL, 303
  - CLK\_SRC\_SPLL\_DIV1, 303
  - CLK\_SRC\_SPLL\_DIV2, 303
  - CLOCK\_DRV\_GetSystemClockSource, 310
  - CLOCK\_DRV\_SetClockSource, 310
  - CLOCK\_DRV\_SetModuleClock, 310
  - CLOCK\_DRV\_SetSystemClock, 312
  - CLOCK\_MANAGER\_CALLBACK\_AFTER, 304
  - CLOCK\_MANAGER\_CALLBACK\_BEFORE, 304
  - CLOCK\_MANAGER\_CALLBACK\_BEFORE\_AFTER, 304
  - CLOCK\_MANAGER\_NOTIFY\_AFTER, 304
  - CLOCK\_MANAGER\_NOTIFY\_BEFORE, 304
  - CLOCK\_MANAGER\_NOTIFY\_RECOVER, 304
  - CLOCK\_MANAGER\_POLICY\_AGREEMENT, 304
  - CLOCK\_MANAGER\_POLICY\_FORCIBLE, 304
  - CLOCK\_SYS\_GetCurrentConfiguration, 312
  - CLOCK\_SYS\_GetErrorCallback, 312
  - CLOCK\_SYS\_GetFreq, 312
  - CLOCK\_SYS\_Init, 312
  - CLOCK\_SYS\_SetConfiguration, 314
  - CLOCK\_SYS\_UpdateConfiguration, 314
  - CLOCK\_TRACE\_SRC\_CORE\_CLK, 304
  - CORE\_CLK\_INDEX, 303
  - clock\_manager\_callback\_t, 303
  - clock\_manager\_callback\_type\_t, 303
  - clock\_manager\_notify\_t, 304
  - clock\_manager\_policy\_t, 304
  - clock\_trace\_src\_t, 304
  - clock\_user\_config\_t, 303
  - DIVIDE\_BY\_EIGHTH, 304
  - DIVIDE\_BY\_FIVE, 304
  - DIVIDE\_BY\_FOUR, 304
  - DIVIDE\_BY\_ONE, 304
  - DIVIDE\_BY\_SEVEN, 304
  - DIVIDE\_BY\_SIX, 304
  - DIVIDE\_BY\_THREE, 304
  - DIVIDE\_BY\_TWO, 304
  - g\_RtcClkInFreq, 315
  - g\_TClkFreq, 315
  - g\_xtal0ClkFreq, 315
  - HSRUN\_MODE, 305
  - MULTIPLY\_BY\_ONE, 305
  - MULTIPLY\_BY\_TWO, 305
  - NO\_MODE, 305
  - NUMBER\_OF\_TCLK\_INPUTS, 303
  - peripheral\_clock\_divider\_t, 304
  - peripheral\_clock\_frac\_t, 304
  - peripheral\_clock\_source\_t, 303
  - peripheralFeaturesList, 315
  - pwr\_modes\_t, 305
  - RUN\_MODE, 305
  - SCG\_ASYNC\_CLOCK\_DISABLE, 305
  - SCG\_ASYNC\_CLOCK\_DIV\_BY\_1, 305
  - SCG\_ASYNC\_CLOCK\_DIV\_BY\_16, 305
  - SCG\_ASYNC\_CLOCK\_DIV\_BY\_2, 305
  - SCG\_ASYNC\_CLOCK\_DIV\_BY\_32, 305
  - SCG\_ASYNC\_CLOCK\_DIV\_BY\_4, 305
  - SCG\_ASYNC\_CLOCK\_DIV\_BY\_64, 305
  - SCG\_ASYNC\_CLOCK\_DIV\_BY\_8, 305
  - SCG\_CLOCKOUT\_SRC\_FIRC, 305
  - SCG\_CLOCKOUT\_SRC\_SCG\_SLOW, 305
  - SCG\_CLOCKOUT\_SRC\_SIRC, 305
  - SCG\_CLOCKOUT\_SRC\_SOSC, 305
  - SCG\_CLOCKOUT\_SRC\_SPLL, 305
  - SCG\_FIRC\_RANGE\_48M, 306
  - SCG\_SIRC\_RANGE\_HIGH, 306
  - SCG\_SOSC\_GAIN\_HIGH, 306
  - SCG\_SOSC\_GAIN\_LOW, 306
  - SCG\_SOSC\_MONITOR\_DISABLE, 306
  - SCG\_SOSC\_MONITOR\_INT, 306
  - SCG\_SOSC\_MONITOR\_RESET, 306
  - SCG\_SOSC\_RANGE\_HIGH, 306
  - SCG\_SOSC\_RANGE\_MID, 306
  - SCG\_SOSC\_REF\_EXT, 306
  - SCG\_SOSC\_REF\_OSC, 306
  - SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_16, 307
  - SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_17, 307
  - SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_18, 307
  - SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_19, 307

- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_20, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_21, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_22, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_23, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_24, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_25, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_26, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_27, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_28, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_29, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_30, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_31, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_32, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_33, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_34, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_35, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_36, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_37, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_38, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_39, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_40, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_41, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_42, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_43, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_44, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_45, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_46, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_47, [307](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_1, [307](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_2, [307](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_3, [307](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_4, [308](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_5, [308](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_6, [308](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_7, [308](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_8, [308](#)
- SCG\_SPLL\_MONITOR\_DISABLE, [308](#)
- SCG\_SPLL\_MONITOR\_INT, [308](#)
- SCG\_SPLL\_MONITOR\_RESET, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_1, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_10, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_11, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_12, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_13, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_14, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_15, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_16, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_2, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_3, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_4, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_5, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_6, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_7, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_8, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_9, [308](#)
- SCG\_SYSTEM\_CLOCK\_SRC\_FIRC, [308](#)
- SCG\_SYSTEM\_CLOCK\_SRC\_NONE, [308](#)
- SCG\_SYSTEM\_CLOCK\_SRC\_SIRC, [308](#)
- SCG\_SYSTEM\_CLOCK\_SRC\_SYS\_OSC, [308](#)
- SIM\_CLKOUT\_DIV\_BY\_1, [309](#)
- SIM\_CLKOUT\_DIV\_BY\_2, [309](#)
- SIM\_CLKOUT\_DIV\_BY\_3, [309](#)
- SIM\_CLKOUT\_DIV\_BY\_4, [309](#)
- SIM\_CLKOUT\_DIV\_BY\_5, [309](#)
- SIM\_CLKOUT\_DIV\_BY\_6, [309](#)
- SIM\_CLKOUT\_DIV\_BY\_7, [309](#)
- SIM\_CLKOUT\_DIV\_BY\_8, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_BUS\_CLK, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_FIRC\_DIV2\_CLK, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_HCLK, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_LPO\_128K\_CLK, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_LPO\_CLK, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_RTC\_CLK, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_SCG\_CLKOUT, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_SIRC\_DIV2\_CLK, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_SOSC\_DIV2\_CLK, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_SPLL\_DIV2\_CLK, [309](#)
- SIM\_LPO\_CLK\_SEL\_LPO\_128K, [309](#)
- SIM\_LPO\_CLK\_SEL\_LPO\_1K, [309](#)
- SIM\_LPO\_CLK\_SEL\_LPO\_32K, [309](#)
- SIM\_LPO\_CLK\_SEL\_NO\_CLOCK, [309](#)
- SIM\_RTCCLK\_SEL\_FIRCDIV1\_CLK, [310](#)
- SIM\_RTCCLK\_SEL\_LPO\_32K, [310](#)
- SIM\_RTCCLK\_SEL\_RTC\_CLKIN, [310](#)
- SIM\_RTCCLK\_SEL\_SOSCDIV1\_CLK, [310](#)
- SLOW\_CLK\_INDEX, [303](#)
- STOP\_MODE, [305](#)
- SYS\_CLK\_MAX\_NO, [303](#)
- scg\_async\_clock\_div\_t, [305](#)
- scg\_clockout\_src\_t, [305](#)
- scg\_firc\_range\_t, [305](#)
- scg\_sirc\_range\_t, [306](#)
- scg\_sosc\_ext\_ref\_t, [306](#)
- scg\_sosc\_gain\_t, [306](#)
- scg\_sosc\_monitor\_mode\_t, [306](#)
- scg\_sosc\_range\_t, [306](#)
- scg\_spll\_clock\_multiply\_t, [306](#)
- scg\_spll\_clock\_prediv\_t, [307](#)
- scg\_spll\_monitor\_mode\_t, [308](#)
- scg\_system\_clock\_div\_t, [308](#)
- scg\_system\_clock\_src\_t, [308](#)
- sim\_clkout\_div\_t, [308](#)
- sim\_clkout\_src\_t, [309](#)
- sim\_lpoclk\_sel\_src\_t, [309](#)
- sim\_rtc\_clk\_sel\_src\_t, [309](#)
- VLPR\_MODE, [305](#)
- VLPS\_MODE, [305](#)
- XOSC\_EXT\_REF, [310](#)
- XOSC\_INT\_OSC, [310](#)
- xosc\_ref\_t, [310](#)

- clock\_manager\_callback\_t
  - Clock Manager Driver, [303](#)
- clock\_manager\_callback\_type\_t
  - Clock Manager Driver, [303](#)
- clock\_manager\_callback\_user\_config\_t, [300](#)
  - callback, [301](#)
  - callbackData, [301](#)
  - callbackType, [301](#)
- clock\_manager\_notify\_t
  - Clock Manager Driver, [304](#)
- clock\_manager\_policy\_t
  - Clock Manager Driver, [304](#)
- clock\_manager\_state\_t, [301](#)
  - callbackConfig, [301](#)
  - callbackNum, [301](#)
  - clockConfigNum, [301](#)
  - configTable, [301](#)
  - curConfigIndex, [301](#)
  - errorCallbackIndex, [301](#)
- clock\_manager\_user\_config\_t, [297](#)
  - pccConfig, [298](#)
  - pmcConfig, [298](#)
  - scgConfig, [298](#)
  - simConfig, [298](#)
- clock\_notify\_struct\_t, [300](#)
  - notifyType, [300](#)
  - policy, [300](#)
  - targetClockConfigIndex, [300](#)
- clock\_source\_config\_t, [299](#)
  - div, [299](#)
  - enable, [299](#)
  - mul, [299](#)
  - outputDiv1, [299](#)
  - outputDiv2, [300](#)
  - refClk, [300](#)
  - refFreq, [300](#)
- clock\_trace\_src\_t
  - Clock Manager Driver, [304](#)
- clock\_user\_config\_t
  - Clock Manager Driver, [303](#)
- clockConfigNum
  - clock\_manager\_state\_t, [301](#)
- clockDivide
  - adc\_converter\_config\_t, [216](#)
  - extension\_adc\_s32k1xx\_t, [236](#)
- clockModeConfig
  - scg\_config\_t, [295](#)
- clockName
  - peripheral\_clock\_config\_t, [296](#)
- clockOutConfig
  - rtc\_init\_config\_t, [896](#)
  - scg\_config\_t, [295](#)
  - sim\_clock\_config\_t, [288](#)
- clockPhase
  - flexio\_spi\_master\_user\_config\_t, [496](#)
  - flexio\_spi\_slave\_user\_config\_t, [497](#)
  - spi\_master\_t, [957](#)
  - spi\_slave\_t, [959](#)
- clockPolarity
  - flexio\_spi\_master\_user\_config\_t, [496](#)
  - flexio\_spi\_slave\_user\_config\_t, [497](#)
  - spi\_master\_t, [957](#)
  - spi\_slave\_t, [959](#)
- clockSelect
  - extension\_ftm\_for\_timer\_t, [998](#)
  - extension\_lptmr\_for\_timer\_t, [998](#)
  - lptmr\_config\_t, [722](#)
  - rtc\_init\_config\_t, [896](#)
- cmc
  - sbc\_can\_ctr\_t, [1017](#)
- cmd
  - csec\_state\_t, [261](#)
- cmdInProgress
  - csec\_state\_t, [261](#)
- cmp\_anmux\_t, [329](#)
  - negativeInputMux, [329](#)
  - negativePortMux, [329](#)
  - positiveInputMux, [330](#)
  - positivePortMux, [330](#)
- cmp\_ch\_list\_t
  - Comparator Driver, [332](#)
- cmp\_ch\_number\_t
  - Comparator Driver, [332](#)
- cmp\_comparator\_t, [328](#)
  - dmaTriggerState, [328](#)
  - filterSampleCount, [328](#)
  - filterSamplePeriod, [328](#)
  - hysteresisLevel, [328](#)
  - inverterState, [328](#)
  - mode, [329](#)
  - outputInterruptTrigger, [329](#)
  - outputSelect, [329](#)
  - pinState, [329](#)
  - powerMode, [329](#)
- cmp\_dac\_t, [330](#)
  - state, [330](#)
  - voltage, [330](#)
  - voltageReferenceSource, [330](#)
- cmp\_fixed\_port\_t
  - Comparator Driver, [332](#)
- cmp\_hysteresis\_t
  - Comparator Driver, [332](#)
- cmp\_inverter\_t
  - Comparator Driver, [333](#)
- cmp\_mode\_t
  - Comparator Driver, [333](#)
- cmp\_module\_t, [331](#)
  - comparator, [331](#)
  - dac, [331](#)
  - mux, [332](#)
  - triggerMode, [332](#)
- cmp\_output\_enable\_t
  - Comparator Driver, [333](#)
- cmp\_output\_select\_t
  - Comparator Driver, [333](#)
- cmp\_output\_trigger\_t

- Comparator Driver, [334](#)
- cmp\_port\_mux\_t
  - Comparator Driver, [334](#)
- cmp\_power\_mode\_t
  - Comparator Driver, [334](#)
- cmp\_trigger\_mode\_t, [330](#)
  - fixedChannel, [331](#)
  - fixedPort, [331](#)
  - programedState, [331](#)
  - roundRobinChannelsState, [331](#)
  - roundRobinInterruptState, [331](#)
  - roundRobinState, [331](#)
  - samples, [331](#)
- cmp\_voltage\_reference\_t
  - Comparator Driver, [334](#)
- cntByte
  - lin\_state\_t, [655](#)
- coll\_resolv\_schd
  - lin\_associate\_frame\_t, [759](#)
- Common Core API., [316](#)
  - BUS\_ACTIVITY\_SET, [316](#)
  - ERROR\_IN\_RESPONSE, [316](#)
  - EVENT\_TRIGGER\_COLLISION\_SET, [316](#)
  - GO\_TO\_SLEEP\_SET, [316](#)
  - OVERRUN, [316](#)
  - SAVE\_CONFIG\_SET, [317](#)
  - SUCCESSFULL\_TRANSFER, [317](#)
- Common Transport Layer API, [318](#)
  - DIAG\_SERVICE\_CALLBACK\_HANDLER, [318](#)
  - GENERAL\_REJECT, [318](#)
  - LD\_ANY\_FUNCTION, [319](#)
  - LD\_ANY\_MESSAGE, [319](#)
  - LD\_ANY\_SUPPLIER, [319](#)
  - LD\_BROADCAST, [319](#)
  - LD\_DATA\_ERROR, [319](#)
  - LD\_FUNCTIONAL\_NAD, [319](#)
  - LD\_LENGTH\_NOT\_CORRECT, [319](#)
  - LD\_LENGTH\_TOO\_SHORT, [319](#)
  - LD\_READ\_OK, [319](#)
  - LD\_SET\_OK, [319](#)
  - LIN\_PRODUCT\_ID, [319](#)
  - LIN\_SERIAL\_NUMBER, [320](#)
  - lin\_diag\_service\_callback, [321](#)
  - NEGATIVE, [320](#)
  - POSITIVE, [320](#)
  - RECEIVING, [320](#)
  - RES\_NEGATIVE, [320](#)
  - RES\_POSITIVE, [320](#)
  - SERVICE\_NOT\_SUPPORTED, [320](#)
  - SERVICE\_TARGET\_RESET, [320](#)
  - SUBFUNCTION\_NOT\_SUPPORTED, [320](#)
  - TRANSMITTING, [320](#)
- compVal1
  - adc\_compare\_config\_t, [218](#)
- compVal2
  - adc\_compare\_config\_t, [218](#)
- comparator
  - cmp\_module\_t, [331](#)
- Comparator (CMP), [322](#)
- Comparator Driver, [326](#)
  - CMP\_AVAILABLE, [333](#)
  - CMP\_BOTH\_EDGES, [334](#)
  - CMP\_CONTINUOUS, [333](#)
  - CMP\_COUT, [334](#)
  - CMP\_COUTA, [334](#)
  - CMP\_DAC, [334](#)
  - CMP\_DISABLED, [333](#)
  - CMP\_DRV\_ClearInputFlags, [335](#)
  - CMP\_DRV\_ClearOutputFlags, [335](#)
  - CMP\_DRV\_ConfigComparator, [335](#)
  - CMP\_DRV\_ConfigDAC, [335](#)
  - CMP\_DRV\_ConfigMUX, [336](#)
  - CMP\_DRV\_ConfigTriggerMode, [336](#)
  - CMP\_DRV\_GetComparatorConfig, [336](#)
  - CMP\_DRV\_GetConfigAll, [337](#)
  - CMP\_DRV\_GetDACConfig, [337](#)
  - CMP\_DRV\_GetDefaultConfig, [337](#)
  - CMP\_DRV\_GetInitConfigAll, [337](#)
  - CMP\_DRV\_GetInitConfigComparator, [338](#)
  - CMP\_DRV\_GetInitConfigDAC, [338](#)
  - CMP\_DRV\_GetInitConfigMUX, [338](#)
  - CMP\_DRV\_GetInitTriggerMode, [339](#)
  - CMP\_DRV\_GetInputFlags, [339](#)
  - CMP\_DRV\_GetMUXConfig, [339](#)
  - CMP\_DRV\_GetOutputFlags, [339](#)
  - CMP\_DRV\_GetTriggerModeConfig, [340](#)
  - CMP\_DRV\_Init, [340](#)
  - CMP\_DRV\_Reset, [340](#)
  - CMP\_FALLING\_EDGE, [334](#)
  - CMP\_HIGH\_SPEED, [334](#)
  - CMP\_INPUT\_FLAGS\_MASK, [332](#)
  - CMP\_INPUT\_FLAGS\_SHIFT, [332](#)
  - CMP\_INVERT, [333](#)
  - CMP\_LEVEL\_HYS\_0, [333](#)
  - CMP\_LEVEL\_HYS\_1, [333](#)
  - CMP\_LEVEL\_HYS\_2, [333](#)
  - CMP\_LEVEL\_HYS\_3, [333](#)
  - CMP\_LOW\_SPEED, [334](#)
  - CMP\_MINUS\_FIXED, [332](#)
  - CMP\_MUX, [334](#)
  - CMP\_NO\_EVENT, [334](#)
  - CMP\_NORMAL, [333](#)
  - CMP\_PLUS\_FIXED, [332](#)
  - CMP\_RISING\_EDGE, [334](#)
  - CMP\_ROUND\_ROBIN\_CHANNELS\_MASK, [332](#)
  - CMP\_ROUND\_ROBIN\_CHANNELS\_SHIFT, [332](#)
  - CMP\_SAMPLED\_FILTERED\_EXT\_CLK, [333](#)
  - CMP\_SAMPLED\_FILTERED\_INT\_CLK, [333](#)
  - CMP\_SAMPLED\_NONFILTERED\_EXT\_CLK, [333](#)
  - CMP\_SAMPLED\_NONFILTERED\_INT\_CLK, [333](#)
  - CMP\_UNAVAILABLE, [333](#)
  - CMP\_VIN1, [334](#)
  - CMP\_VIN2, [334](#)
  - CMP\_WINDOWED, [333](#)
  - CMP\_WINDOWED\_FILTERED, [333](#)
  - CMP\_WINDOWED\_RESAMPLED, [333](#)

- cmp\_ch\_list\_t, 332
- cmp\_ch\_number\_t, 332
- cmp\_fixed\_port\_t, 332
- cmp\_hysteresis\_t, 332
- cmp\_inverter\_t, 333
- cmp\_mode\_t, 333
- cmp\_output\_enable\_t, 333
- cmp\_output\_select\_t, 333
- cmp\_output\_trigger\_t, 334
- cmp\_port\_mux\_t, 334
- cmp\_power\_mode\_t, 334
- cmp\_voltage\_reference\_t, 334
- compareEnable
  - adc\_compare\_config\_t, 218
- compareGreaterThanEnable
  - adc\_compare\_config\_t, 218
- compareHigh
  - ewm\_init\_config\_t, 408
- compareLow
  - ewm\_init\_config\_t, 408
- compareRangeFuncEnable
  - adc\_compare\_config\_t, 218
- compareValue
  - lptmr\_config\_t, 722
- comparedValue
  - ftm\_output\_cmp\_ch\_param\_t, 567
  - oc\_output\_ch\_param\_t, 838
- compensation
  - rtc\_init\_config\_t, 896
- compensationInterval
  - rtc\_init\_config\_t, 896
- complementChecksum
  - crc\_user\_config\_t, 251
- complementaryChannelPolarity
  - pwm\_channel\_t, 888
- configTable
  - clock\_manager\_state\_t, 301
- configs
  - power\_manager\_state\_t, 868
- configsNumber
  - power\_manager\_state\_t, 868
- configured\_NAD\_ptr
  - lin\_node\_attribute\_t, 758
- continuousConvEn
  - adc\_group\_config\_t, 234
- continuousConvEnable
  - adc\_converter\_config\_t, 216
- continuousModeEn
  - channel\_extension\_ftm\_for\_ic\_t, 616
  - ftm\_input\_ch\_param\_t, 556
- continuousModeEnable
  - pdb\_timer\_config\_t, 849
- control
  - sbc\_factories\_conf\_t, 1021
- controlRegisterLock
  - rtc\_register\_lock\_config\_t, 899
- Controller Area Network - Peripheral Abstraction Layer (CAN PAL), 342
- CAN\_AbortTransfer, 352
- CAN\_CLK\_SOURCE\_OSC, 351
- CAN\_CLK\_SOURCE\_PERIPH, 351
- CAN\_ConfigRemoteResponseBuff, 352
- CAN\_ConfigRxBuff, 353
- CAN\_ConfigTxBuff, 353
- CAN\_DISABLE\_MODE, 352
- CAN\_Deinit, 354
- CAN\_FD\_DATA\_BITRATE, 351
- CAN\_GetBitrate, 354
- CAN\_GetDefaultConfig, 354
- CAN\_GetTransferStatus, 354
- CAN\_Init, 355
- CAN\_InstallEventCallback, 355
- CAN\_LOOPBACK\_MODE, 352
- CAN\_MSG\_ID\_EXT, 352
- CAN\_MSG\_ID\_STD, 352
- CAN\_NOMINAL\_BITRATE, 351
- CAN\_NORMAL\_MODE, 352
- CAN\_PAYLOAD\_SIZE\_16, 351
- CAN\_PAYLOAD\_SIZE\_32, 352
- CAN\_PAYLOAD\_SIZE\_64, 352
- CAN\_PAYLOAD\_SIZE\_8, 351
- CAN\_Receive, 355
- CAN\_ReceiveBlocking, 356
- CAN\_Send, 356
- CAN\_SendBlocking, 357
- CAN\_SetBitrate, 357
- CAN\_SetRxFilter, 357
- can\_bitrate\_phase\_t, 351
- can\_clk\_source\_t, 351
- can\_fd\_payload\_size\_t, 351
- can\_msg\_id\_type\_t, 352
- can\_operation\_modes\_t, 352
- Controller Area Network with Flexible Data Rate (FlexCAN), 359
- Cooked API, 361
  - ld\_receive\_message, 361
  - ld\_rx\_status, 361
  - ld\_send\_message, 362
  - ld\_tx\_status, 362
- coscs
  - sbc\_trans\_stat\_t, 1023
- count
  - pcc\_config\_t, 296
- counter
  - ftm\_quad\_decoder\_state\_t, 591
- counterDirection
  - ftm\_quad\_decoder\_state\_t, 591
- counterUnits
  - lptmr\_config\_t, 722
- cpnc
  - sbc\_can\_ctr\_t, 1017
- cpncCanPartNetCfg
  - uja113x\_init\_cfg\_t, 145
- cpnerr
  - sbc\_trans\_stat\_t, 1023
- cpns



- sbc\_trans\_stat\_t, [1023](#)
- crc\_spec\_t, [208](#)
  - init, [208](#)
  - poly, [208](#)
  - refIn, [208](#)
  - refOut, [209](#)
  - xorOut, [209](#)
- crc\_transpose\_t
  - CRC Driver, [251](#)
- crc\_user\_config\_t, [250](#)
  - complementChecksum, [251](#)
  - seed, [251](#)
  - writeTranspose, [251](#)
- Cryptographic Services Engine (CSEc), [363](#)
- cs
  - can\_message\_t, [349](#)
  - flexcan\_msgbuff\_t, [447](#)
- csec\_boot\_flavor\_t
  - CSEc Driver, [263](#)
- csec\_call\_sequence\_t
  - CSEc Driver, [263](#)
- csec\_cmd\_t
  - CSEc Driver, [264](#)
- csec\_key\_id\_t
  - CSEc Driver, [264](#)
- csec\_state\_t, [260](#)
  - callback, [261](#)
  - callbackParam, [261](#)
  - cmd, [261](#)
  - cmdInProgress, [261](#)
  - errCode, [261](#)
  - fullSize, [261](#)
  - index, [261](#)
  - inputBuff, [261](#)
  - iv, [261](#)
  - keyId, [261](#)
  - mac, [261](#)
  - macLen, [261](#)
  - macWritten, [262](#)
  - msgLen, [262](#)
  - outputBuff, [262](#)
  - partSize, [262](#)
  - seq, [262](#)
  - verifStatus, [262](#)
- csec\_status\_t
  - CSEc Driver, [263](#)
- cts
  - sbc\_trans\_stat\_t, [1024](#)
- curConfigIndex
  - clock\_manager\_state\_t, [301](#)
- current\_id
  - lin\_protocol\_state\_t, [769](#)
- currentConfig
  - power\_manager\_state\_t, [868](#)
- currentEventId
  - lin\_state\_t, [655](#)
- currentId
  - lin\_state\_t, [655](#)
- currentNodeState
  - lin\_state\_t, [655](#)
- currentPid
  - lin\_state\_t, [655](#)
- cw
  - sbc\_trans\_evnt\_stat\_t, [1026](#)
- cwe
  - sbc\_trans\_evnt\_t, [1017](#)
- Cyclic Redundancy Check (CRC), [364](#)
- DADDR
  - edma\_software\_tcd\_t, [382](#)
- DAYS\_IN\_A\_LEAP\_YEAR
  - RTC Driver, [899](#)
- DAYS\_IN\_A\_YEAR
  - RTC Driver, [899](#)
- DFLASH\_IFR\_READRESOURCE\_ADDRESS
  - Flash Memory (Flash), [421](#)
- DFlashBase
  - Flash Memory (Flash), [434](#), [435](#)
- DFlashSize
  - Flash Memory (Flash), [435](#)
- DIAG\_INTERLEAVE\_MODE
  - Low level API, [774](#)
- DIAG\_NO\_RESPONSE
  - Low level API, [774](#)
- DIAG\_NONE
  - Low level API, [774](#)
- DIAG\_NOT\_START
  - Low level API, [774](#)
- DIAG\_ONLY\_MODE
  - Low level API, [774](#)
- DIAG\_RESPONSE
  - Low level API, [774](#)
- DIAG\_SERVICE\_CALLBACK\_HANDLER
  - Common Transport Layer API, [318](#)
- DID1\_ADDRESS
  - Register address map, [920](#)
- DID2\_ADDRESS
  - Register address map, [920](#)
- DIVIDE\_BY\_EIGHTH
  - Clock Manager Driver, [304](#)
- DIVIDE\_BY\_FIVE
  - Clock Manager Driver, [304](#)
- DIVIDE\_BY\_FOUR
  - Clock Manager Driver, [304](#)
- DIVIDE\_BY\_ONE
  - Clock Manager Driver, [304](#)
- DIVIDE\_BY\_SEVEN
  - Clock Manager Driver, [304](#)
- DIVIDE\_BY\_SIX
  - Clock Manager Driver, [304](#)
- DIVIDE\_BY\_THREE
  - Clock Manager Driver, [304](#)
- DIVIDE\_BY\_TWO
  - Clock Manager Driver, [304](#)
- DLAST\_SGA
  - edma\_software\_tcd\_t, [382](#)
- DLC\_0

- 01. CAN & CAN partial networking configuration, [156](#)
- DLC\_1
  - 01. CAN & CAN partial networking configuration, [156](#)
- DLC\_2
  - 01. CAN & CAN partial networking configuration, [156](#)
- DLC\_3
  - 01. CAN & CAN partial networking configuration, [156](#)
- DLC\_4
  - 01. CAN & CAN partial networking configuration, [156](#)
- DLC\_5
  - 01. CAN & CAN partial networking configuration, [156](#)
- DLC\_6
  - 01. CAN & CAN partial networking configuration, [156](#)
- DLC\_7
  - 01. CAN & CAN partial networking configuration, [156](#)
- DLC\_8
  - 01. CAN & CAN partial networking configuration, [156](#)
- DM0\_ADDRESS
  - Register address map, [920](#)
- DM1\_ADDRESS
  - Register address map, [920](#)
- DM2\_ADDRESS
  - Register address map, [920](#)
- DM3\_ADDRESS
  - Register address map, [921](#)
- DM4\_ADDRESS
  - Register address map, [921](#)
- DM5\_ADDRESS
  - Register address map, [921](#)
- DM6\_ADDRESS
  - Register address map, [921](#)
- DM7\_ADDRESS
  - Register address map, [921](#)
- DOFF
  - edma\_software\_tcd\_t, [383](#)
- DR\_ADDRESS
  - Register address map, [921](#)
- dac
  - cmp\_module\_t, [331](#)
- datRate
  - sbc\_can\_conf\_t, [1018](#)
- data
  - can\_message\_t, [349](#)
  - flexcan\_msgbuff\_t, [447](#)
- data\_length
  - flexcan\_data\_info\_t, [449](#)
- dataBitrate
  - can\_user\_config\_t, [350](#)
- dataLen
  - flexcan\_msgbuff\_t, [447](#)
- dataLenCode
  - uja113x\_cpnc\_cfg\_t, [152](#)
- dataMask
  - eim\_user\_channel\_config\_t, [398](#)
  - sbc\_can\_conf\_t, [1018](#)
- dataPin
  - flexio\_uart\_user\_config\_t, [509](#)
- dataPinRx
  - extension\_flexio\_for\_uart\_t, [1053](#)
- dataPinTx
  - extension\_flexio\_for\_uart\_t, [1053](#)
- day
  - rtc\_timedate\_t, [895](#)
- deadTime
  - ftm\_combined\_ch\_param\_t, [580](#)
  - ftm\_independent\_ch\_param\_t, [579](#)
- deadTimePrescaler
  - ftm\_pwm\_param\_t, [582](#)
- deadTimeValue
  - ftm\_pwm\_param\_t, [582](#)
- deadtime
  - pwm\_channel\_t, [888](#)
- deadtimePrescaler
  - pwm\_ftm\_timebase\_t, [887](#)
- debug
  - wdg\_option\_mode\_t, [1062](#)
  - wdog\_op\_mode\_t, [1071](#)
- DefaultISR
  - Interrupt Manager (Interrupt), [638](#)
- delay\_integer
  - lin\_schedule\_data\_t, [761](#)
- delayArray
  - adc\_group\_config\_t, [234](#)
- delayType
  - adc\_group\_config\_t, [235](#)
- destAddr
  - edma\_transfer\_config\_t, [380](#)
- destLastAddrAdjust
  - edma\_transfer\_config\_t, [380](#)
- destModulo
  - edma\_transfer\_config\_t, [381](#)
- destOffset
  - edma\_transfer\_config\_t, [381](#)
- destTransferSize
  - edma\_transfer\_config\_t, [381](#)
- diag\_IO\_control
  - Diagnostic services, [368](#)
- diag\_clear\_flag
  - Diagnostic services, [367](#)
- diag\_fault\_memory\_clear
  - Diagnostic services, [367](#)
- diag\_fault\_memory\_read
  - Diagnostic services, [367](#)
- diag\_get\_flag
  - Diagnostic services, [368](#)
- diag\_interleave\_state
  - lin\_tl\_descriptor\_t, [763](#)



- diag\_interleaved\_state\_t
  - Low level API, [773](#)
- diag\_read\_data\_by\_identifier
  - Diagnostic services, [368](#)
- diag\_session\_control
  - Diagnostic services, [369](#)
- diag\_state
  - lin\_tl\_descriptor\_t, [763](#)
- diag\_write\_data\_by\_identifier
  - Diagnostic services, [369](#)
- Diagnostic services, [366](#)
  - diag\_IO\_control, [368](#)
  - diag\_clear\_flag, [367](#)
  - diag\_fault\_memory\_clear, [367](#)
  - diag\_fault\_memory\_read, [367](#)
  - diag\_get\_flag, [368](#)
  - diag\_read\_data\_by\_identifier, [368](#)
  - diag\_session\_control, [369](#)
  - diag\_write\_data\_by\_identifier, [369](#)
- diagnostic\_class
  - lin\_protocol\_user\_config\_t, [766](#)
- diagnostic\_mode
  - lin\_protocol\_state\_t, [769](#)
- direction
  - flexio\_uart\_user\_config\_t, [509](#)
  - pin\_settings\_config\_t, [858](#)
- div
  - clock\_source\_config\_t, [299](#)
  - module\_clk\_config\_t, [298](#)
- div1
  - scg\_firc\_config\_t, [291](#)
  - scg\_sirc\_config\_t, [290](#)
  - scg\_sosc\_config\_t, [289](#)
  - scg\_spill\_config\_t, [292](#)
- div2
  - scg\_firc\_config\_t, [291](#)
  - scg\_sirc\_config\_t, [290](#)
  - scg\_sosc\_config\_t, [289](#)
  - scg\_spill\_config\_t, [292](#)
- divBus
  - scg\_system\_clock\_config\_t, [288](#)
- divCore
  - scg\_system\_clock\_config\_t, [288](#)
- divEnable
  - sim\_trace\_clock\_config\_t, [287](#)
- divFraction
  - sim\_trace\_clock\_config\_t, [287](#)
- divSlow
  - scg\_system\_clock\_config\_t, [288](#)
- divider
  - peripheral\_clock\_config\_t, [296](#)
  - sim\_clock\_out\_config\_t, [284](#)
  - sim\_trace\_clock\_config\_t, [287](#)
- dividers
  - sys\_clk\_config\_t, [299](#)
- dlc
  - sbc\_frame\_t, [1018](#)
- dm0
  - uja113x\_dm\_cfg\_t, [153](#)
- dm1
  - uja113x\_dm\_cfg\_t, [153](#)
- dm2
  - uja113x\_dm\_cfg\_t, [153](#)
- dm3
  - uja113x\_dm\_cfg\_t, [153](#)
- dm4
  - uja113x\_dm\_cfg\_t, [154](#)
- dm5
  - uja113x\_dm\_cfg\_t, [154](#)
- dm6
  - uja113x\_dm\_cfg\_t, [154](#)
- dm7
  - uja113x\_dm\_cfg\_t, [154](#)
- dmCanDataMaskCfg
  - uja113x\_init\_cfg\_t, [145](#)
- dmaChannel
  - flexio\_uart\_user\_config\_t, [509](#)
  - i2c\_slave\_t, [626](#)
  - lpi2c\_master\_user\_config\_t, [674](#)
  - lpi2c\_slave\_user\_config\_t, [675](#)
- dmaChannel1
  - i2c\_master\_t, [625](#)
- dmaChannel2
  - i2c\_master\_t, [625](#)
- dmaEnable
  - adc\_converter\_config\_t, [216](#)
  - pdb\_timer\_config\_t, [849](#)
- dmaRequest
  - lptmr\_config\_t, [722](#)
- dmaTriggerState
  - cmp\_comparator\_t, [328](#)
- Driver and cluster management, [370](#)
  - l\_sys\_init, [370](#)
- driverType
  - flexio\_i2c\_master\_user\_config\_t, [468](#)
  - flexio\_i2s\_master\_user\_config\_t, [478](#)
  - flexio\_i2s\_slave\_user\_config\_t, [479](#)
  - flexio\_spi\_master\_user\_config\_t, [496](#)
  - flexio\_spi\_slave\_user\_config\_t, [497](#)
  - flexio\_uart\_user\_config\_t, [509](#)
- drv\_config\_t, [1082](#)
  - isInit, [1083](#)
  - lpspiIntace, [1083](#)
  - watchdogCtr, [1083](#)
- dstOffsetEnable
  - edma\_loop\_transfer\_config\_t, [379](#)
- dummy
  - lpspi\_state\_t, [706](#)
- duty
  - pwm\_channel\_t, [888](#)
- EDMA Driver, [371](#)
  - EDMA\_ARBITRATION\_FIXED\_PRIORITY, [384](#)
  - EDMA\_ARBITRATION\_ROUND\_ROBIN, [384](#)
  - EDMA\_CHN\_DEFAULT\_PRIORITY, [384](#)
  - EDMA\_CHN\_ERR\_INT, [384](#)
  - EDMA\_CHN\_ERROR, [385](#)

- EDMA\_CHN\_HALF\_MAJOR\_LOOP\_INT, [384](#)
- EDMA\_CHN\_MAJOR\_LOOP\_INT, [384](#)
- EDMA\_CHN\_NORMAL, [385](#)
- EDMA\_CHN\_PRIORITY\_0, [384](#)
- EDMA\_CHN\_PRIORITY\_1, [384](#)
- EDMA\_CHN\_PRIORITY\_10, [384](#)
- EDMA\_CHN\_PRIORITY\_11, [384](#)
- EDMA\_CHN\_PRIORITY\_12, [384](#)
- EDMA\_CHN\_PRIORITY\_13, [384](#)
- EDMA\_CHN\_PRIORITY\_14, [384](#)
- EDMA\_CHN\_PRIORITY\_15, [384](#)
- EDMA\_CHN\_PRIORITY\_2, [384](#)
- EDMA\_CHN\_PRIORITY\_3, [384](#)
- EDMA\_CHN\_PRIORITY\_4, [384](#)
- EDMA\_CHN\_PRIORITY\_5, [384](#)
- EDMA\_CHN\_PRIORITY\_6, [384](#)
- EDMA\_CHN\_PRIORITY\_7, [384](#)
- EDMA\_CHN\_PRIORITY\_8, [384](#)
- EDMA\_CHN\_PRIORITY\_9, [384](#)
- EDMA\_DRV\_CancelTransfer, [386](#)
- EDMA\_DRV\_ChannelInit, [386](#)
- EDMA\_DRV\_ClearTCD, [387](#)
- EDMA\_DRV\_ConfigLoopTransfer, [387](#)
- EDMA\_DRV\_ConfigMultiBlockTransfer, [387](#)
- EDMA\_DRV\_ConfigScatterGatherTransfer, [388](#)
- EDMA\_DRV\_ConfigSingleBlockTransfer, [389](#)
- EDMA\_DRV\_ConfigureInterrupt, [389](#)
- EDMA\_DRV\_Deinit, [389](#)
- EDMA\_DRV\_DisableRequestsOnTransfer↔  
Complete, [389](#)
- EDMA\_DRV\_GetChannelStatus, [390](#)
- EDMA\_DRV\_GetRemainingMajorIterationsCount,  
[390](#)
- EDMA\_DRV\_Init, [390](#)
- EDMA\_DRV\_InstallCallback, [391](#)
- EDMA\_DRV\_PushConfigToReg, [391](#)
- EDMA\_DRV\_PushConfigToSTCD, [391](#)
- EDMA\_DRV\_ReleaseChannel, [392](#)
- EDMA\_DRV\_SetChannelRequestAndTrigger, [392](#)
- EDMA\_DRV\_SetDestAddr, [392](#)
- EDMA\_DRV\_SetDestLastAddrAdjustment, [392](#)
- EDMA\_DRV\_SetDestOffset, [393](#)
- EDMA\_DRV\_SetDestWriteChunkSize, [393](#)
- EDMA\_DRV\_SetMajorLoopIterationCount, [393](#)
- EDMA\_DRV\_SetMinorLoopBlockSize, [393](#)
- EDMA\_DRV\_SetScatterGatherLink, [393](#)
- EDMA\_DRV\_SetSrcAddr, [394](#)
- EDMA\_DRV\_SetSrcLastAddrAdjustment, [394](#)
- EDMA\_DRV\_SetSrcOffset, [394](#)
- EDMA\_DRV\_SetSrcReadChunkSize, [394](#)
- EDMA\_DRV\_StartChannel, [394](#)
- EDMA\_DRV\_StopChannel, [395](#)
- EDMA\_DRV\_TriggerSwRequest, [395](#)
- EDMA\_ERR\_LSB\_MASK, [383](#)
- EDMA\_MODULO\_128B, [385](#)
- EDMA\_MODULO\_128KB, [385](#)
- EDMA\_MODULO\_128MB, [385](#)
- EDMA\_MODULO\_16B, [385](#)
- EDMA\_MODULO\_16KB, [385](#)
- EDMA\_MODULO\_16MB, [385](#)
- EDMA\_MODULO\_1GB, [385](#)
- EDMA\_MODULO\_1KB, [385](#)
- EDMA\_MODULO\_1MB, [385](#)
- EDMA\_MODULO\_256B, [385](#)
- EDMA\_MODULO\_256KB, [385](#)
- EDMA\_MODULO\_256MB, [385](#)
- EDMA\_MODULO\_2B, [385](#)
- EDMA\_MODULO\_2GB, [385](#)
- EDMA\_MODULO\_2KB, [385](#)
- EDMA\_MODULO\_2MB, [385](#)
- EDMA\_MODULO\_32B, [385](#)
- EDMA\_MODULO\_32KB, [385](#)
- EDMA\_MODULO\_32MB, [385](#)
- EDMA\_MODULO\_4B, [385](#)
- EDMA\_MODULO\_4KB, [385](#)
- EDMA\_MODULO\_4MB, [385](#)
- EDMA\_MODULO\_512B, [385](#)
- EDMA\_MODULO\_512KB, [385](#)
- EDMA\_MODULO\_512MB, [385](#)
- EDMA\_MODULO\_64B, [385](#)
- EDMA\_MODULO\_64KB, [385](#)
- EDMA\_MODULO\_64MB, [385](#)
- EDMA\_MODULO\_8B, [385](#)
- EDMA\_MODULO\_8KB, [385](#)
- EDMA\_MODULO\_8MB, [385](#)
- EDMA\_MODULO\_OFF, [385](#)
- EDMA\_TRANSFER\_MEM2MEM, [386](#)
- EDMA\_TRANSFER\_MEM2PERIPH, [386](#)
- EDMA\_TRANSFER\_PERIPH2MEM, [386](#)
- EDMA\_TRANSFER\_PERIPH2PERIPH, [386](#)
- EDMA\_TRANSFER\_SIZE\_1B, [386](#)
- EDMA\_TRANSFER\_SIZE\_2B, [386](#)
- EDMA\_TRANSFER\_SIZE\_4B, [386](#)
- edma\_arbitration\_algorithm\_t, [383](#)
- edma\_callback\_t, [383](#)
- edma\_channel\_interrupt\_t, [384](#)
- edma\_channel\_priority\_t, [384](#)
- edma\_chn\_status\_t, [384](#)
- edma\_modulo\_t, [385](#)
- edma\_transfer\_size\_t, [385](#)
- edma\_transfer\_type\_t, [386](#)
- STCD\_ADDR, [383](#)
- STCD\_SIZE, [383](#)
- EDMA\_ARBITRATION\_FIXED\_PRIORITY  
EDMA Driver, [384](#)
- EDMA\_ARBITRATION\_ROUND\_ROBIN  
EDMA Driver, [384](#)
- EDMA\_CHN\_DEFAULT\_PRIORITY  
EDMA Driver, [384](#)
- EDMA\_CHN\_ERR\_INT  
EDMA Driver, [384](#)
- EDMA\_CHN\_ERROR  
EDMA Driver, [385](#)
- EDMA\_CHN\_HALF\_MAJOR\_LOOP\_INT  
EDMA Driver, [384](#)
- EDMA\_CHN\_MAJOR\_LOOP\_INT

- EDMA Driver, [384](#)
- EDMA\_CHN\_NORMAL
  - EDMA Driver, [385](#)
- EDMA\_CHN\_PRIORITY\_0
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_1
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_10
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_11
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_12
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_13
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_14
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_15
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_2
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_3
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_4
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_5
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_6
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_7
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_8
  - EDMA Driver, [384](#)
- EDMA\_CHN\_PRIORITY\_9
  - EDMA Driver, [384](#)
- EDMA\_DRV\_CancelTransfer
  - EDMA Driver, [386](#)
- EDMA\_DRV\_ChannelInit
  - EDMA Driver, [386](#)
- EDMA\_DRV\_ClearTCD
  - EDMA Driver, [387](#)
- EDMA\_DRV\_ConfigLoopTransfer
  - EDMA Driver, [387](#)
- EDMA\_DRV\_ConfigMultiBlockTransfer
  - EDMA Driver, [387](#)
- EDMA\_DRV\_ConfigScatterGatherTransfer
  - EDMA Driver, [388](#)
- EDMA\_DRV\_ConfigSingleBlockTransfer
  - EDMA Driver, [389](#)
- EDMA\_DRV\_ConfigureInterrupt
  - EDMA Driver, [389](#)
- EDMA\_DRV\_Deinit
  - EDMA Driver, [389](#)
- EDMA\_DRV\_DisableRequestsOnTransferComplete
  - EDMA Driver, [389](#)
- EDMA\_DRV\_GetChannelStatus
  - EDMA Driver, [390](#)
- EDMA\_DRV\_GetRemainingMajorIterationsCount
  - EDMA Driver, [390](#)
- EDMA\_DRV\_Init
  - EDMA Driver, [390](#)
- EDMA\_DRV\_InstallCallback
  - EDMA Driver, [391](#)
- EDMA\_DRV\_PushConfigToReg
  - EDMA Driver, [391](#)
- EDMA\_DRV\_PushConfigToSTCD
  - EDMA Driver, [391](#)
- EDMA\_DRV\_ReleaseChannel
  - EDMA Driver, [392](#)
- EDMA\_DRV\_SetChannelRequestAndTrigger
  - EDMA Driver, [392](#)
- EDMA\_DRV\_SetDestAddr
  - EDMA Driver, [392](#)
- EDMA\_DRV\_SetDestLastAddrAdjustment
  - EDMA Driver, [392](#)
- EDMA\_DRV\_SetDestOffset
  - EDMA Driver, [393](#)
- EDMA\_DRV\_SetDestWriteChunkSize
  - EDMA Driver, [393](#)
- EDMA\_DRV\_SetMajorLoopIterationCount
  - EDMA Driver, [393](#)
- EDMA\_DRV\_SetMinorLoopBlockSize
  - EDMA Driver, [393](#)
- EDMA\_DRV\_SetScatterGatherLink
  - EDMA Driver, [393](#)
- EDMA\_DRV\_SetSrcAddr
  - EDMA Driver, [394](#)
- EDMA\_DRV\_SetSrcLastAddrAdjustment
  - EDMA Driver, [394](#)
- EDMA\_DRV\_SetSrcOffset
  - EDMA Driver, [394](#)
- EDMA\_DRV\_SetSrcReadChunkSize
  - EDMA Driver, [394](#)
- EDMA\_DRV\_StartChannel
  - EDMA Driver, [394](#)
- EDMA\_DRV\_StopChannel
  - EDMA Driver, [395](#)
- EDMA\_DRV\_TriggerSwRequest
  - EDMA Driver, [395](#)
- EDMA\_ERR\_LSB\_MASK
  - EDMA Driver, [383](#)
- EDMA\_MODULO\_128B
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_128KB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_128MB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_16B
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_16KB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_16MB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_1GB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_1KB

- EDMA Driver, [385](#)
- EDMA\_MODULO\_1MB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_256B
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_256KB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_256MB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_2B
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_2GB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_2KB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_2MB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_32B
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_32KB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_32MB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_4B
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_4KB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_4MB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_512B
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_512KB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_512MB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_64B
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_64KB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_64MB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_8B
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_8KB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_8MB
  - EDMA Driver, [385](#)
- EDMA\_MODULO\_OFF
  - EDMA Driver, [385](#)
- EDMA\_TRANSFER\_MEM2MEM
  - EDMA Driver, [386](#)
- EDMA\_TRANSFER\_MEM2PERIPH
  - EDMA Driver, [386](#)
- EDMA\_TRANSFER\_PERIPH2MEM
  - EDMA Driver, [386](#)
- EDMA\_TRANSFER\_PERIPH2PERIPH
  - EDMA Driver, [386](#)
- EDMA\_TRANSFER\_SIZE\_1B
  - EDMA Driver, [386](#)
- EDMA\_TRANSFER\_SIZE\_2B
  - EDMA Driver, [386](#)
- EDMA\_TRANSFER\_SIZE\_4B
  - EDMA Driver, [386](#)
- EEE\_COMPLETE\_INTERRUPT\_QUICK\_WRITE
  - Flash Memory (Flash), [425](#)
- EEE\_DISABLE
  - Flash Memory (Flash), [425](#)
- EEE\_ENABLE
  - Flash Memory (Flash), [425](#)
- EEE\_QUICK\_WRITE
  - Flash Memory (Flash), [425](#)
- EEE\_STATUS\_QUERY
  - Flash Memory (Flash), [425](#)
- EEESize
  - Flash Memory (Flash), [435](#)
- EERAMBase
  - Flash Memory (Flash), [435](#)
- EIM Driver, [396](#)
  - EIM\_CHECKBITMASK\_DEFAULT, [398](#)
  - EIM\_DATAMASK\_DEFAULT, [398](#)
  - EIM\_DRV\_ConfigChannel, [399](#)
  - EIM\_DRV\_Deinit, [399](#)
  - EIM\_DRV\_GetChannelConfig, [399](#)
  - EIM\_DRV\_GetDefaultConfig, [399](#)
  - EIM\_DRV\_Init, [400](#)
- EIM\_CHECKBITMASK\_DEFAULT
  - EIM Driver, [398](#)
- EIM\_DATAMASK\_DEFAULT
  - EIM Driver, [398](#)
- EIM\_DRV\_ConfigChannel
  - EIM Driver, [399](#)
- EIM\_DRV\_Deinit
  - EIM Driver, [399](#)
- EIM\_DRV\_GetChannelConfig
  - EIM Driver, [399](#)
- EIM\_DRV\_GetDefaultConfig
  - EIM Driver, [399](#)
- EIM\_DRV\_Init
  - EIM Driver, [400](#)
- ENC\_HIGH\_N
  - 03. Fail-safe configuration, [180](#)
- ENC\_HIGH\_NRS
  - 03. Fail-safe configuration, [180](#)
- ENC\_LOW
  - 03. Fail-safe configuration, [180](#)
- ENC\_TIMER
  - 03. Fail-safe configuration, [180](#)
- ENSC\_FULL
  - 03. Fail-safe configuration, [180](#)
- ENSC\_NONE
  - 03. Fail-safe configuration, [180](#)
- ENSC\_OVERVOLTAGE
  - 03. Fail-safe configuration, [180](#)
- ENSC\_UNDRVOLTAGE
  - 03. Fail-safe configuration, [180](#)
- ERM Driver, [401](#)

- ERM\_DRV\_ClearEvent, [404](#)
- ERM\_DRV\_Deinit, [404](#)
- ERM\_DRV\_GetErrorDetail, [404](#)
- ERM\_DRV\_GetInterruptConfig, [404](#)
- ERM\_DRV\_Init, [405](#)
- ERM\_DRV\_SetInterruptConfig, [405](#)
- ERM\_EVENT\_NON\_CORRECTABLE, [404](#)
- ERM\_EVENT\_NONE, [404](#)
- ERM\_EVENT\_SINGLE\_BIT, [404](#)
- erm\_ecc\_event\_t, [404](#)
- ERM\_DRV\_ClearEvent
  - ERM Driver, [404](#)
- ERM\_DRV\_Deinit
  - ERM Driver, [404](#)
- ERM\_DRV\_GetErrorDetail
  - ERM Driver, [404](#)
- ERM\_DRV\_GetInterruptConfig
  - ERM Driver, [404](#)
- ERM\_DRV\_Init
  - ERM Driver, [405](#)
- ERM\_DRV\_SetInterruptConfig
  - ERM Driver, [405](#)
- ERM\_EVENT\_NON\_CORRECTABLE
  - ERM Driver, [404](#)
- ERM\_EVENT\_NONE
  - ERM Driver, [404](#)
- ERM\_EVENT\_SINGLE\_BIT
  - ERM Driver, [404](#)
- ERROR\_IN\_RESPONSE
  - Common Core API., [316](#)
- EVENT\_TRIGGER\_COLLISION\_SET
  - Common Core API., [316](#)
- EWM Driver, [406](#)
  - EWM\_DRV\_GetDefaultConfig, [409](#)
  - EWM\_DRV\_GetInputPinAssertLogic, [409](#)
  - EWM\_DRV\_Init, [409](#)
  - EWM\_DRV\_Refresh, [410](#)
  - EWM\_IN\_ASSERT\_DISABLED, [409](#)
  - EWM\_IN\_ASSERT\_ON\_LOGIC\_ONE, [409](#)
  - EWM\_IN\_ASSERT\_ON\_LOGIC\_ZERO, [409](#)
  - ewm\_in\_assert\_logic\_t, [409](#)
- EWM\_DRV\_GetDefaultConfig
  - EWM Driver, [409](#)
- EWM\_DRV\_GetInputPinAssertLogic
  - EWM Driver, [409](#)
- EWM\_DRV\_Init
  - EWM Driver, [409](#)
- EWM\_DRV\_Refresh
  - EWM Driver, [410](#)
- EWM\_IN\_ASSERT\_DISABLED
  - EWM Driver, [409](#)
- EWM\_IN\_ASSERT\_ON\_LOGIC\_ONE
  - EWM Driver, [409](#)
- EWM\_IN\_ASSERT\_ON\_LOGIC\_ZERO
  - EWM Driver, [409](#)
- eccs
  - sbc\_mtpnv\_stat\_t, [1028](#)
- edgeAlignement
  - ftm\_input\_ch\_param\_t, [556](#)
- edma\_arbitration\_algorithm\_t
  - EDMA Driver, [383](#)
- edma\_callback\_t
  - EDMA Driver, [383](#)
- edma\_channel\_config\_t, [377](#)
  - callback, [378](#)
  - callbackParam, [378](#)
  - channelPriority, [378](#)
  - enableTrigger, [378](#)
  - virtChnConfig, [378](#)
- edma\_channel\_interrupt\_t
  - EDMA Driver, [384](#)
- edma\_channel\_priority\_t
  - EDMA Driver, [384](#)
- edma\_chn\_state\_t, [377](#)
  - callback, [377](#)
  - parameter, [377](#)
  - status, [377](#)
  - virtChn, [377](#)
- edma\_chn\_status\_t
  - EDMA Driver, [384](#)
- edma\_loop\_transfer\_config\_t, [379](#)
  - dstOffsetEnable, [379](#)
  - majorLoopChnLinkEnable, [379](#)
  - majorLoopChnLinkNumber, [379](#)
  - majorLoopIterationCount, [379](#)
  - minorLoopChnLinkEnable, [380](#)
  - minorLoopChnLinkNumber, [380](#)
  - minorLoopOffset, [380](#)
  - srcOffsetEnable, [380](#)
- edma\_modulo\_t
  - EDMA Driver, [385](#)
- edma\_scatter\_gather\_list\_t, [378](#)
  - address, [378](#)
  - length, [378](#)
  - type, [378](#)
- edma\_software\_tcd\_t, [382](#)
  - ATTR, [382](#)
  - BITER, [382](#)
  - CITER, [382](#)
  - CSR, [382](#)
  - DADDR, [382](#)
  - DLAST\_SGA, [382](#)
  - DOFF, [383](#)
  - NBYTES, [383](#)
  - SADDR, [383](#)
  - SLAST, [383](#)
  - SOFF, [383](#)
- edma\_state\_t, [379](#)
  - virtChnState, [379](#)
- edma\_transfer\_config\_t, [380](#)
  - destAddr, [380](#)
  - destLastAddrAdjust, [380](#)
  - destModulo, [381](#)
  - destOffset, [381](#)
  - destTransferSize, [381](#)
  - interruptEnable, [381](#)

- loopTransferConfig, [381](#)
- minorByteTransferCount, [381](#)
- scatterGatherEnable, [381](#)
- scatterGatherNextDescAddr, [381](#)
- srcAddr, [381](#)
- srcLastAddrAdjust, [381](#)
- srcModulo, [382](#)
- srcOffset, [382](#)
- srcTransferSize, [382](#)
- edma\_transfer\_size\_t
  - EDMA Driver, [385](#)
- edma\_transfer\_type\_t
  - EDMA Driver, [386](#)
- edma\_user\_config\_t, [376](#)
  - chnArbitration, [377](#)
  - haltOnError, [377](#)
- eim\_user\_channel\_config\_t, [398](#)
  - channel, [398](#)
  - checkBitMask, [398](#)
  - dataMask, [398](#)
  - enable, [398](#)
- enCtrl
  - uja113x\_fsc\_cfg\_t, [179](#)
- enDriverCtrl
  - uja113x\_fsc\_cfg\_t, [179](#)
- enShutdownCtrl
  - uja113x\_fsc\_cfg\_t, [179](#)
- enable
  - clock\_source\_config\_t, [299](#)
  - eim\_user\_channel\_config\_t, [398](#)
  - pmc\_lpo\_clock\_config\_t, [297](#)
  - sim\_clock\_out\_config\_t, [284](#)
- enableBRS
  - can\_buff\_config\_t, [348](#)
- enableComplementaryChannel
  - pwm\_channel\_t, [888](#)
- enableDma
  - sim\_plat\_gate\_config\_t, [286](#)
- enableEim
  - sim\_plat\_gate\_config\_t, [286](#)
- enableErm
  - sim\_plat\_gate\_config\_t, [286](#)
- enableExternalTrigger
  - ftm\_combined\_ch\_param\_t, [580](#)
  - ftm\_independent\_ch\_param\_t, [579](#)
  - ftm\_output\_cmp\_ch\_param\_t, [567](#)
- enableExternalTriggerOnNextChn
  - ftm\_combined\_ch\_param\_t, [580](#)
- enableFD
  - can\_buff\_config\_t, [348](#)
  - can\_user\_config\_t, [350](#)
- enableInLowPower
  - scg\_firc\_config\_t, [291](#)
  - scg\_sirc\_config\_t, [290](#)
  - scg\_sosc\_config\_t, [289](#)
- enableInStop
  - scg\_firc\_config\_t, [291](#)
  - scg\_sirc\_config\_t, [291](#)
- scg\_sosc\_config\_t, [289](#)
- scg\_spill\_config\_t, [292](#)
- enableInitializationTrigger
  - ftm\_user\_config\_t, [524](#)
- enableLpo1k
  - sim\_lpo\_clock\_config\_t, [285](#)
- enableLpo32k
  - sim\_lpo\_clock\_config\_t, [285](#)
- enableModifiedCombine
  - ftm\_combined\_ch\_param\_t, [580](#)
- enableMpu
  - sim\_plat\_gate\_config\_t, [286](#)
- enableMscm
  - sim\_plat\_gate\_config\_t, [286](#)
- enableNonCorrectable
  - erm\_interrupt\_config\_t, [403](#)
- enableNotification
  - ftm\_state\_t, [522](#)
- enableQspiRefClk
  - sim\_qspi\_ref\_clk\_gating\_t, [286](#)
- enableReloadOnTrigger
  - lpit\_user\_channel\_config\_t, [690](#)
- enableRunInDebug
  - lpit\_user\_config\_t, [690](#)
- enableRunInDoze
  - lpit\_user\_config\_t, [690](#)
- enableSecondChannelOutput
  - ftm\_combined\_ch\_param\_t, [581](#)
  - ftm\_independent\_ch\_param\_t, [579](#)
- enableSingleCorrection
  - erm\_interrupt\_config\_t, [403](#)
- enableStartOnTrigger
  - lpit\_user\_channel\_config\_t, [690](#)
- enableStopOnInterrupt
  - lpit\_user\_channel\_config\_t, [691](#)
- enableTrigger
  - edma\_channel\_config\_t, [378](#)
- endAddr
  - mpu\_region\_config\_t, [802](#)
  - mpu\_user\_config\_t, [791](#)
- Enhanced Direct Memory Access (eDMA), [411](#)
- erm\_ecc\_event\_t
  - ERM Driver, [404](#)
- erm\_interrupt\_config\_t, [403](#)
  - enableNonCorrectable, [403](#)
  - enableSingleCorrection, [403](#)
- erm\_user\_config\_t, [403](#)
  - channel, [403](#)
  - interruptCfg, [403](#)
- errCode
  - csec\_state\_t, [261](#)
- Error Injection Module (EIM), [412](#)
- Error Reporting Module (ERM), [414](#)
- error\_callback
  - FlexCANState, [448](#)
- error\_in\_res
  - lin\_word\_status\_str\_t, [756](#)
- error\_in\_response



- lin\_protocol\_state\_t, 769
- errorCallbackIndex
  - clock\_manager\_state\_t, 301
  - power\_manager\_state\_t, 868
- errorCallbackParam
  - FlexCANState, 448
- event\_trigger\_collision\_flg
  - lin\_master\_data\_t, 768
  - lin\_word\_status\_str\_t, 756
- events
  - sbc\_status\_group\_t, 1028
- ewm\_in\_assert\_logic\_t
  - EWM Driver, 409
- ewm\_init\_config\_t, 408
  - assertLogic, 408
  - compareHigh, 408
  - compareLow, 408
  - interruptEnable, 408
  - prescaler, 408
- extPinSrc
  - sim\_tclk\_config\_t, 285
- extRef
  - scg\_sosc\_config\_t, 289
- extension
  - adc\_config\_t, 236
  - can\_user\_config\_t, 350
  - i2c\_master\_t, 625
  - i2s\_user\_config\_t, 605
  - ic\_config\_t, 616
  - mpu\_region\_config\_t, 802
  - oc\_config\_t, 838
  - spi\_master\_t, 957
  - spi\_slave\_t, 959
  - timer\_config\_t, 997
  - uart\_user\_config\_t, 1052
  - wdg\_config\_t, 1063
- extension\_adc\_s32k1xx\_t, 236
  - clockDivide, 236
  - inputClock, 236
  - pdbPrescaler, 236
  - resolution, 237
  - supplyMonitoringEnable, 237
  - voltageRef, 237
- extension\_ewm\_for\_wdg\_t, 1062
  - assertLogic, 1062
  - prescalerValue, 1062
- extension\_flexcan\_rx\_fifo\_t, 350
  - idFilterTable, 351
  - idFormat, 351
  - numIdFilters, 351
- extension\_flexio\_for\_i2c\_t, 624
  - sclPin, 624
  - sdaPin, 624
- extension\_flexio\_for\_i2s\_t, 605
  - rxPin, 605
  - sckPin, 606
  - txPin, 606
  - wsPin, 606
- extension\_flexio\_for\_spi\_t, 959
  - misoPin, 960
  - mosiPin, 960
  - sckPin, 960
  - ssPin, 960
- extension\_flexio\_for\_uart\_t, 1053
  - dataPinRx, 1053
  - dataPinTx, 1053
- extension\_ftm\_for\_ic\_t, 616
  - ftmClockSource, 617
  - ftmPrescaler, 617
- extension\_ftm\_for\_oc\_t, 839
  - ftmClockSource, 839
  - ftmPrescaler, 839
  - maxCountValue, 839
- extension\_ftm\_for\_timer\_t, 998
  - clockSelect, 998
  - finalValue, 998
  - prescaler, 998
- extension\_lptmr\_for\_timer\_t, 997
  - bypassPrescaler, 998
  - clockSelect, 998
  - prescaler, 998
- External Watchdog Monitor (EWM), 416
- FC\_ADDRESS
  - Register address map, 921
- FF\_pdu\_received
  - lin\_tl\_descriptor\_t, 763
- FLASH\_CALLBACK\_CS
  - Flash Memory (Flash), 421
- FLASH\_DRV\_CheckSum
  - Flash Memory (Flash), 425
- FLASH\_DRV\_ClearReadColisionFlag
  - Flash Memory (Flash), 426
- FLASH\_DRV\_DisableCmdCompleteInterupt
  - Flash Memory (Flash), 426
- FLASH\_DRV\_DisableReadColisionInterupt
  - Flash Memory (Flash), 426
- FLASH\_DRV\_EnableCmdCompleteInterupt
  - Flash Memory (Flash), 426
- FLASH\_DRV\_EnableReadColisionInterupt
  - Flash Memory (Flash), 426
- FLASH\_DRV\_EraseAllBlock
  - Flash Memory (Flash), 427
- FLASH\_DRV\_EraseResume
  - Flash Memory (Flash), 427
- FLASH\_DRV\_EraseSector
  - Flash Memory (Flash), 427
- FLASH\_DRV\_EraseSuspend
  - Flash Memory (Flash), 428
- FLASH\_DRV\_GetCmdCompleteFlag
  - Flash Memory (Flash), 428
- FLASH\_DRV\_GetDefaultConfig
  - Flash Memory (Flash), 429
- FLASH\_DRV\_GetPFlashProtection
  - Flash Memory (Flash), 429
- FLASH\_DRV\_GetReadColisionFlag
  - Flash Memory (Flash), 429

- FLASH\_DRV\_GetSecurityState  
Flash Memory (Flash), [429](#)
- FLASH\_DRV\_Init  
Flash Memory (Flash), [430](#)
- FLASH\_DRV\_Program  
Flash Memory (Flash), [430](#)
- FLASH\_DRV\_ProgramCheck  
Flash Memory (Flash), [430](#)
- FLASH\_DRV\_ProgramOnce  
Flash Memory (Flash), [431](#)
- FLASH\_DRV\_ReadOnce  
Flash Memory (Flash), [431](#)
- FLASH\_DRV\_SecurityBypass  
Flash Memory (Flash), [432](#)
- FLASH\_DRV\_SetPFlashProtection  
Flash Memory (Flash), [432](#)
- FLASH\_DRV\_VerifyAllBlock  
Flash Memory (Flash), [433](#)
- FLASH\_DRV\_VerifySection  
Flash Memory (Flash), [433](#)
- FLASH\_NOT\_SECURE  
Flash Memory (Flash), [421](#)
- FLASH\_SECURE\_BACKDOOR\_DISABLED  
Flash Memory (Flash), [422](#)
- FLASH\_SECURE\_BACKDOOR\_ENABLED  
Flash Memory (Flash), [422](#)
- FLASH\_SECURITY\_STATE\_KEYEN  
Flash Memory (Flash), [422](#)
- FLASH\_SECURITY\_STATE\_UNSECURED  
Flash Memory (Flash), [422](#)
- FLEXCAN\_DISABLE\_MODE  
FlexCAN Driver, [452](#)
- FLEXCAN\_DRV\_AbortTransfer  
FlexCAN Driver, [454](#)
- FLEXCAN\_DRV\_ConfigRemoteResponseMb  
FlexCAN Driver, [454](#)
- FLEXCAN\_DRV\_ConfigRxFifo  
FlexCAN Driver, [454](#)
- FLEXCAN\_DRV\_ConfigRxMb  
FlexCAN Driver, [455](#)
- FLEXCAN\_DRV\_ConfigTxMb  
FlexCAN Driver, [455](#)
- FLEXCAN\_DRV\_Deinit  
FlexCAN Driver, [455](#)
- FLEXCAN\_DRV\_GetBitrate  
FlexCAN Driver, [455](#)
- FLEXCAN\_DRV\_GetDefaultConfig  
FlexCAN Driver, [456](#)
- FLEXCAN\_DRV\_GetErrorStatus  
FlexCAN Driver, [456](#)
- FLEXCAN\_DRV\_GetTransferStatus  
FlexCAN Driver, [456](#)
- FLEXCAN\_DRV\_Init  
FlexCAN Driver, [457](#)
- FLEXCAN\_DRV\_InstallErrorCallback  
FlexCAN Driver, [457](#)
- FLEXCAN\_DRV\_InstallEventCallback  
FlexCAN Driver, [457](#)
- FLEXCAN\_DRV\_Receive  
FlexCAN Driver, [457](#)
- FLEXCAN\_DRV\_ReceiveBlocking  
FlexCAN Driver, [458](#)
- FLEXCAN\_DRV\_RxFifo  
FlexCAN Driver, [458](#)
- FLEXCAN\_DRV\_RxFifoBlocking  
FlexCAN Driver, [458](#)
- FLEXCAN\_DRV\_Send  
FlexCAN Driver, [459](#)
- FLEXCAN\_DRV\_SendBlocking  
FlexCAN Driver, [459](#)
- FLEXCAN\_DRV\_SetBitrate  
FlexCAN Driver, [459](#)
- FLEXCAN\_DRV\_SetRxFifoGlobalMask  
FlexCAN Driver, [460](#)
- FLEXCAN\_DRV\_SetRxIndividualMask  
FlexCAN Driver, [460](#)
- FLEXCAN\_DRV\_SetRxMaskType  
FlexCAN Driver, [460](#)
- FLEXCAN\_DRV\_SetRxMb14Mask  
FlexCAN Driver, [460](#)
- FLEXCAN\_DRV\_SetRxMb15Mask  
FlexCAN Driver, [461](#)
- FLEXCAN\_DRV\_SetRxMbGlobalMask  
FlexCAN Driver, [461](#)
- FLEXCAN\_EVENT\_ERROR  
FlexCAN Driver, [452](#)
- FLEXCAN\_EVENT\_RX\_COMPLETE  
FlexCAN Driver, [452](#)
- FLEXCAN\_EVENT\_RXFIFO\_COMPLETE  
FlexCAN Driver, [452](#)
- FLEXCAN\_EVENT\_RXFIFO\_OVERFLOW  
FlexCAN Driver, [452](#)
- FLEXCAN\_EVENT\_RXFIFO\_WARNING  
FlexCAN Driver, [452](#)
- FLEXCAN\_EVENT\_TX\_COMPLETE  
FlexCAN Driver, [452](#)
- FLEXCAN\_FREEZE\_MODE  
FlexCAN Driver, [452](#)
- FLEXCAN\_LISTEN\_ONLY\_MODE  
FlexCAN Driver, [452](#)
- FLEXCAN\_LOOPBACK\_MODE  
FlexCAN Driver, [452](#)
- FLEXCAN\_MB\_IDLE  
FlexCAN Driver, [452](#)
- FLEXCAN\_MB\_RX\_BUSY  
FlexCAN Driver, [452](#)
- FLEXCAN\_MB\_TX\_BUSY  
FlexCAN Driver, [452](#)
- FLEXCAN\_MSG\_ID\_EXT  
FlexCAN Driver, [452](#)
- FLEXCAN\_MSG\_ID\_STD  
FlexCAN Driver, [452](#)
- FLEXCAN\_NORMAL\_MODE  
FlexCAN Driver, [452](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_104  
FlexCAN Driver, [453](#)



- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_112
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_120
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_128
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_16
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_24
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_32
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_40
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_48
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_56
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_64
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_72
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_8
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_80
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_88
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_96
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FORMAT\_A
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FORMAT\_B
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FORMAT\_C
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_FIFO\_ID\_FORMAT\_D
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_MASK\_GLOBAL
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RX\_MASK\_INDIVIDUAL
  - FlexCAN Driver, [453](#)
- FLEXCAN\_RXFIFO\_USING\_INTERRUPTS
  - FlexCAN Driver, [453](#)
- FLEXIO\_DRIVER\_TYPE\_DMA
  - FlexIO Common Driver, [462](#)
- FLEXIO\_DRIVER\_TYPE\_INTERRUPTS
  - FlexIO Common Driver, [462](#)
- FLEXIO\_DRIVER\_TYPE\_POLLING
  - FlexIO Common Driver, [462](#)
- FLEXIO\_DRV\_DeinitDevice
  - FlexIO Common Driver, [462](#)
- FLEXIO\_DRV\_InitDevice
  - FlexIO Common Driver, [464](#)
- FLEXIO\_DRV\_Reset
  - FlexIO Common Driver, [464](#)
- FLEXIO\_I2C\_DRV\_GenerateNineClock
  - FlexIO I2C Driver, [469](#)
- FLEXIO\_I2C\_DRV\_GetBusStatus
  - FlexIO I2C Driver, [469](#)
- FLEXIO\_I2C\_DRV\_GetDefaultConfig
  - FlexIO I2C Driver, [469](#)
- FLEXIO\_I2C\_DRV\_MasterDeinit
  - FlexIO I2C Driver, [470](#)
- FLEXIO\_I2C\_DRV\_MasterGetBaudRate
  - FlexIO I2C Driver, [470](#)
- FLEXIO\_I2C\_DRV\_MasterGetStatus
  - FlexIO I2C Driver, [470](#)
- FLEXIO\_I2C\_DRV\_MasterInit
  - FlexIO I2C Driver, [470](#)
- FLEXIO\_I2C\_DRV\_MasterReceiveData
  - FlexIO I2C Driver, [471](#)
- FLEXIO\_I2C\_DRV\_MasterReceiveDataBlocking
  - FlexIO I2C Driver, [471](#)
- FLEXIO\_I2C\_DRV\_MasterSendData
  - FlexIO I2C Driver, [471](#)
- FLEXIO\_I2C\_DRV\_MasterSendDataBlocking
  - FlexIO I2C Driver, [472](#)
- FLEXIO\_I2C\_DRV\_MasterSetBaudRate
  - FlexIO I2C Driver, [472](#)
- FLEXIO\_I2C\_DRV\_MasterSetSlaveAddr
  - FlexIO I2C Driver, [473](#)
- FLEXIO\_I2C\_DRV\_MasterTransferAbort
  - FlexIO I2C Driver, [473](#)
- FLEXIO\_I2C\_DRV\_StatusGenerateNineClock
  - FlexIO I2C Driver, [473](#)
- FLEXIO\_I2S\_DRV\_MasterDeinit
  - FlexIO I2S Driver, [480](#)
- FLEXIO\_I2S\_DRV\_MasterGetBaudRate
  - FlexIO I2S Driver, [480](#)
- FLEXIO\_I2S\_DRV\_MasterGetDefaultConfig
  - FlexIO I2S Driver, [481](#)
- FLEXIO\_I2S\_DRV\_MasterGetStatus
  - FlexIO I2S Driver, [481](#)
- FLEXIO\_I2S\_DRV\_MasterInit
  - FlexIO I2S Driver, [481](#)
- FLEXIO\_I2S\_DRV\_MasterReceiveData
  - FlexIO I2S Driver, [482](#)
- FLEXIO\_I2S\_DRV\_MasterReceiveDataBlocking
  - FlexIO I2S Driver, [482](#)
- FLEXIO\_I2S\_DRV\_MasterSendData
  - FlexIO I2S Driver, [482](#)
- FLEXIO\_I2S\_DRV\_MasterSendDataBlocking
  - FlexIO I2S Driver, [483](#)
- FLEXIO\_I2S\_DRV\_MasterSetConfig
  - FlexIO I2S Driver, [483](#)
- FLEXIO\_I2S\_DRV\_MasterSetRxBuffer
  - FlexIO I2S Driver, [483](#)
- FLEXIO\_I2S\_DRV\_MasterSetTxBuffer
  - FlexIO I2S Driver, [484](#)
- FLEXIO\_I2S\_DRV\_MasterTransferAbort
  - FlexIO I2S Driver, [484](#)
- FLEXIO\_I2S\_DRV\_SlaveDeinit
  - FlexIO I2S Driver, [484](#)
- FLEXIO\_I2S\_DRV\_SlaveGetDefaultConfig
  - FlexIO I2S Driver, [485](#)

- FLEXIO\_I2S\_DRV\_SlaveGetStatus
  - FlexIO I2S Driver, [485](#)
- FLEXIO\_I2S\_DRV\_SlaveInit
  - FlexIO I2S Driver, [485](#)
- FLEXIO\_I2S\_DRV\_SlaveReceiveData
  - FlexIO I2S Driver, [486](#)
- FLEXIO\_I2S\_DRV\_SlaveReceiveDataBlocking
  - FlexIO I2S Driver, [486](#)
- FLEXIO\_I2S\_DRV\_SlaveSendData
  - FlexIO I2S Driver, [487](#)
- FLEXIO\_I2S\_DRV\_SlaveSendDataBlocking
  - FlexIO I2S Driver, [487](#)
- FLEXIO\_I2S\_DRV\_SlaveSetConfig
  - FlexIO I2S Driver, [488](#)
- FLEXIO\_I2S\_DRV\_SlaveSetRxBuffer
  - FlexIO I2S Driver, [488](#)
- FLEXIO\_I2S\_DRV\_SlaveSetTxBuffer
  - FlexIO I2S Driver, [489](#)
- FLEXIO\_I2S\_DRV\_SlaveTransferAbort
  - FlexIO I2S Driver, [489](#)
- FLEXIO\_SPI\_DRV\_MasterDeinit
  - FlexIO SPI Driver, [499](#)
- FLEXIO\_SPI\_DRV\_MasterGetBaudRate
  - FlexIO SPI Driver, [499](#)
- FLEXIO\_SPI\_DRV\_MasterGetDefaultConfig
  - FlexIO SPI Driver, [500](#)
- FLEXIO\_SPI\_DRV\_MasterGetStatus
  - FlexIO SPI Driver, [500](#)
- FLEXIO\_SPI\_DRV\_MasterInit
  - FlexIO SPI Driver, [500](#)
- FLEXIO\_SPI\_DRV\_MasterSetBaudRate
  - FlexIO SPI Driver, [500](#)
- FLEXIO\_SPI\_DRV\_MasterTransfer
  - FlexIO SPI Driver, [501](#)
- FLEXIO\_SPI\_DRV\_MasterTransferAbort
  - FlexIO SPI Driver, [501](#)
- FLEXIO\_SPI\_DRV\_MasterTransferBlocking
  - FlexIO SPI Driver, [501](#)
- FLEXIO\_SPI\_DRV\_SlaveDeinit
  - FlexIO SPI Driver, [502](#)
- FLEXIO\_SPI\_DRV\_SlaveGetDefaultConfig
  - FlexIO SPI Driver, [502](#)
- FLEXIO\_SPI\_DRV\_SlaveGetStatus
  - FlexIO SPI Driver, [502](#)
- FLEXIO\_SPI\_DRV\_SlaveInit
  - FlexIO SPI Driver, [503](#)
- FLEXIO\_SPI\_DRV\_SlaveTransfer
  - FlexIO SPI Driver, [503](#)
- FLEXIO\_SPI\_DRV\_SlaveTransferAbort
  - FlexIO SPI Driver, [504](#)
- FLEXIO\_SPI\_DRV\_SlaveTransferBlocking
  - FlexIO SPI Driver, [504](#)
- FLEXIO\_SPI\_TRANSFER\_1BYTE
  - FlexIO SPI Driver, [499](#)
- FLEXIO\_SPI\_TRANSFER\_2BYTE
  - FlexIO SPI Driver, [499](#)
- FLEXIO\_SPI\_TRANSFER\_4BYTE
  - FlexIO SPI Driver, [499](#)
- FLEXIO\_SPI\_TRANSFER\_LSB\_FIRST
  - FlexIO SPI Driver, [499](#)
- FLEXIO\_SPI\_TRANSFER\_MSB\_FIRST
  - FlexIO SPI Driver, [499](#)
- FLEXIO\_UART\_DIRECTION\_RX
  - FlexIO UART Driver, [510](#)
- FLEXIO\_UART\_DIRECTION\_TX
  - FlexIO UART Driver, [510](#)
- FLEXIO\_UART\_DRV\_Deinit
  - FlexIO UART Driver, [510](#)
- FLEXIO\_UART\_DRV\_GetBaudRate
  - FlexIO UART Driver, [510](#)
- FLEXIO\_UART\_DRV\_GetDefaultConfig
  - FlexIO UART Driver, [510](#)
- FLEXIO\_UART\_DRV\_GetStatus
  - FlexIO UART Driver, [511](#)
- FLEXIO\_UART\_DRV\_Init
  - FlexIO UART Driver, [511](#)
- FLEXIO\_UART\_DRV\_ReceiveData
  - FlexIO UART Driver, [511](#)
- FLEXIO\_UART\_DRV\_ReceiveDataBlocking
  - FlexIO UART Driver, [512](#)
- FLEXIO\_UART\_DRV\_SendData
  - FlexIO UART Driver, [512](#)
- FLEXIO\_UART\_DRV\_SendDataBlocking
  - FlexIO UART Driver, [512](#)
- FLEXIO\_UART\_DRV\_SetConfig
  - FlexIO UART Driver, [513](#)
- FLEXIO\_UART\_DRV\_SetRxBuffer
  - FlexIO UART Driver, [513](#)
- FLEXIO\_UART\_DRV\_SetTxBuffer
  - FlexIO UART Driver, [513](#)
- FLEXIO\_UART\_DRV\_TransferAbort
  - FlexIO UART Driver, [514](#)
- FNMC\_DISABLE
  - SBC configuration control options (0x74), [930](#)
- FNMC\_ENABLE
  - SBC configuration control options (0x74), [930](#)
- FSC\_ADDRESS
  - Register address map, [921](#)
- FTFx\_DPHRASE\_SIZE
  - Flash Memory (Flash), [422](#)
- FTFx\_ERASE\_ALL\_BLOCK
  - Flash Memory (Flash), [422](#)
- FTFx\_ERASE\_ALL\_BLOCK\_UNSECURE
  - Flash Memory (Flash), [422](#)
- FTFx\_ERASE\_BLOCK
  - Flash Memory (Flash), [422](#)
- FTFx\_ERASE\_SECTOR
  - Flash Memory (Flash), [422](#)
- FTFx\_FSTAT\_ERROR\_BITS
  - Flash Memory (Flash), [422](#)
- FTFx\_LONGWORD\_SIZE
  - Flash Memory (Flash), [422](#)
- FTFx\_PFLASH\_SWAP
  - Flash Memory (Flash), [422](#)
- FTFx\_PHRASE\_SIZE
  - Flash Memory (Flash), [422](#)

- FTFx\_PROGRAM\_CHECK
  - Flash Memory (Flash), [422](#)
- FTFx\_PROGRAM\_LONGWORD
  - Flash Memory (Flash), [423](#)
- FTFx\_PROGRAM\_ONCE
  - Flash Memory (Flash), [423](#)
- FTFx\_PROGRAM\_PARTITION
  - Flash Memory (Flash), [423](#)
- FTFx\_PROGRAM\_PHRASE
  - Flash Memory (Flash), [423](#)
- FTFx\_PROGRAM\_SECTION
  - Flash Memory (Flash), [423](#)
- FTFx\_READ\_ONCE
  - Flash Memory (Flash), [423](#)
- FTFx\_READ\_RESOURCE
  - Flash Memory (Flash), [423](#)
- FTFx\_RSRC\_CODE\_REG
  - Flash Memory (Flash), [423](#)
- FTFx\_SECURITY\_BY\_PASS
  - Flash Memory (Flash), [423](#)
- FTFx\_SET\_EERAM
  - Flash Memory (Flash), [423](#)
- FTFx\_SWAP\_COMPLETE
  - Flash Memory (Flash), [423](#)
- FTFx\_SWAP\_READY
  - Flash Memory (Flash), [423](#)
- FTFx\_SWAP\_REPORT\_STATUS
  - Flash Memory (Flash), [423](#)
- FTFx\_SWAP\_SET\_IN\_COMPLETE
  - Flash Memory (Flash), [423](#)
- FTFx\_SWAP\_SET\_IN\_PREPARE
  - Flash Memory (Flash), [424](#)
- FTFx\_SWAP\_SET\_INDICATOR\_ADDR
  - Flash Memory (Flash), [424](#)
- FTFx\_SWAP\_UNINIT
  - Flash Memory (Flash), [424](#)
- FTFx\_SWAP\_UPDATE
  - Flash Memory (Flash), [424](#)
- FTFx\_SWAP\_UPDATE\_ERASED
  - Flash Memory (Flash), [424](#)
- FTFx\_VERIFY\_ALL\_BLOCK
  - Flash Memory (Flash), [424](#)
- FTFx\_VERIFY\_BLOCK
  - Flash Memory (Flash), [424](#)
- FTFx\_VERIFY\_SECTION
  - Flash Memory (Flash), [424](#)
- FTFx\_WORD\_SIZE
  - Flash Memory (Flash), [424](#)
- FTM\_ABSOLUTE\_VALUE
  - FlexTimer Output Compare Driver (FTM\_OC), [568](#)
- FTM\_BDM\_MODE\_00
  - FlexTimer (FTM), [528](#)
- FTM\_BDM\_MODE\_01
  - FlexTimer (FTM), [528](#)
- FTM\_BDM\_MODE\_10
  - FlexTimer (FTM), [528](#)
- FTM\_BDM\_MODE\_11
  - FlexTimer (FTM), [528](#)
- FTM\_BOTH\_EDGES
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_CHANNEL0\_FLAG
  - FlexTimer (FTM), [530](#)
- FTM\_CHANNEL0\_INT\_ENABLE
  - FlexTimer (FTM), [529](#)
- FTM\_CHANNEL1\_FLAG
  - FlexTimer (FTM), [530](#)
- FTM\_CHANNEL1\_INT\_ENABLE
  - FlexTimer (FTM), [529](#)
- FTM\_CHANNEL2\_FLAG
  - FlexTimer (FTM), [530](#)
- FTM\_CHANNEL2\_INT\_ENABLE
  - FlexTimer (FTM), [529](#)
- FTM\_CHANNEL3\_FLAG
  - FlexTimer (FTM), [530](#)
- FTM\_CHANNEL3\_INT\_ENABLE
  - FlexTimer (FTM), [529](#)
- FTM\_CHANNEL4\_FLAG
  - FlexTimer (FTM), [530](#)
- FTM\_CHANNEL4\_INT\_ENABLE
  - FlexTimer (FTM), [529](#)
- FTM\_CHANNEL5\_FLAG
  - FlexTimer (FTM), [530](#)
- FTM\_CHANNEL5\_INT\_ENABLE
  - FlexTimer (FTM), [529](#)
- FTM\_CHANNEL6\_FLAG
  - FlexTimer (FTM), [530](#)
- FTM\_CHANNEL6\_INT\_ENABLE
  - FlexTimer (FTM), [529](#)
- FTM\_CHANNEL7\_FLAG
  - FlexTimer (FTM), [530](#)
- FTM\_CHANNEL7\_INT\_ENABLE
  - FlexTimer (FTM), [529](#)
- FTM\_CHANNEL\_TRIGGER\_FLAG
  - FlexTimer (FTM), [530](#)
- FTM\_CLEAR\_ON\_MATCH
  - FlexTimer Output Compare Driver (FTM\_OC), [568](#)
- FTM\_CLOCK\_DIVID\_BY\_1
  - FlexTimer (FTM), [528](#)
- FTM\_CLOCK\_DIVID\_BY\_128
  - FlexTimer (FTM), [528](#)
- FTM\_CLOCK\_DIVID\_BY\_16
  - FlexTimer (FTM), [528](#)
- FTM\_CLOCK\_DIVID\_BY\_2
  - FlexTimer (FTM), [528](#)
- FTM\_CLOCK\_DIVID\_BY\_32
  - FlexTimer (FTM), [528](#)
- FTM\_CLOCK\_DIVID\_BY\_4
  - FlexTimer (FTM), [528](#)
- FTM\_CLOCK\_DIVID\_BY\_64
  - FlexTimer (FTM), [528](#)
- FTM\_CLOCK\_DIVID\_BY\_8
  - FlexTimer (FTM), [528](#)
- FTM\_CLOCK\_SOURCE\_EXTERNALCLK
  - FlexTimer (FTM), [528](#)
- FTM\_CLOCK\_SOURCE\_FIXEDCLK
  - FlexTimer (FTM), [528](#)

- FTM\_CLOCK\_SOURCE\_NONE
  - FlexTimer (FTM), [528](#)
- FTM\_CLOCK\_SOURCE\_SYSTEMCLK
  - FlexTimer (FTM), [528](#)
- FTM\_DEADTIME\_DIVID\_BY\_1
  - FlexTimer (FTM), [529](#)
- FTM\_DEADTIME\_DIVID\_BY\_16
  - FlexTimer (FTM), [529](#)
- FTM\_DEADTIME\_DIVID\_BY\_4
  - FlexTimer (FTM), [529](#)
- FTM\_DISABLE\_OPERATION
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_DISABLE\_OUTPUT
  - FlexTimer Output Compare Driver (FTM\_OC), [568](#)
- FTM\_DRV\_ClearChSC
  - FlexTimer (FTM), [531](#)
- FTM\_DRV\_ClearChnEventStatus
  - FlexTimer (FTM), [530](#)
- FTM\_DRV\_ClearFaultFlagDetected
  - FlexTimer (FTM), [531](#)
- FTM\_DRV\_ClearStatusFlags
  - FlexTimer (FTM), [531](#)
- FTM\_DRV\_ControlChannelOutput
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔PWM), [584](#)
- FTM\_DRV\_ConvertFreqToPeriodTicks
  - FlexTimer (FTM), [531](#)
- FTM\_DRV\_CounterRead
  - FlexTimer Module Counter Driver (FTM\_MC), [563](#)
- FTM\_DRV\_CounterReset
  - FlexTimer (FTM), [531](#)
- FTM\_DRV\_CounterStart
  - FlexTimer Module Counter Driver (FTM\_MC), [563](#)
- FTM\_DRV\_CounterStop
  - FlexTimer Module Counter Driver (FTM\_MC), [563](#)
- FTM\_DRV\_Deinit
  - FlexTimer (FTM), [532](#)
- FTM\_DRV\_DeinitInputCapture
  - FlexTimer Input Capture Driver (FTM\_IC), [558](#)
- FTM\_DRV\_DeinitOutputCompare
  - FlexTimer Output Compare Driver (FTM\_OC), [568](#)
- FTM\_DRV\_DeinitPwm
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔PWM), [584](#)
- FTM\_DRV\_DisableFaultInt
  - FlexTimer (FTM), [532](#)
- FTM\_DRV\_DisableInterrupts
  - FlexTimer (FTM), [532](#)
- FTM\_DRV\_EnableInterrupts
  - FlexTimer (FTM), [532](#)
- FTM\_DRV\_FastUpdatePwmChannels
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔PWM), [584](#)
- FTM\_DRV\_GenerateHardwareTrigger
  - FlexTimer (FTM), [533](#)
- FTM\_DRV\_GetChInputState
  - FlexTimer (FTM), [533](#)
- FTM\_DRV\_GetChOutputValue
  - FlexTimer (FTM), [534](#)
- FTM\_DRV\_GetChnCountVal
  - FlexTimer (FTM), [533](#)
- FTM\_DRV\_GetChnEdgeLevel
  - FlexTimer (FTM), [533](#)
- FTM\_DRV\_GetChnEventStatus
  - FlexTimer (FTM), [534](#)
- FTM\_DRV\_GetClockFilterPs
  - FlexTimer (FTM), [534](#)
- FTM\_DRV\_GetCounter
  - FlexTimer (FTM), [535](#)
- FTM\_DRV\_GetCounterInitVal
  - FlexTimer (FTM), [535](#)
- FTM\_DRV\_GetDefaultConfig
  - FlexTimer (FTM), [535](#)
- FTM\_DRV\_GetEnabledInterrupts
  - FlexTimer (FTM), [535](#)
- FTM\_DRV\_GetEventStatus
  - FlexTimer (FTM), [536](#)
- FTM\_DRV\_GetFrequency
  - FlexTimer (FTM), [536](#)
- FTM\_DRV\_GetInputCaptureMeasurement
  - FlexTimer Input Capture Driver (FTM\_IC), [558](#)
- FTM\_DRV\_GetMod
  - FlexTimer (FTM), [536](#)
- FTM\_DRV\_GetStatusFlags
  - FlexTimer (FTM), [536](#)
- FTM\_DRV\_GetTriggerControlled
  - FlexTimer (FTM), [537](#)
- FTM\_DRV\_Init
  - FlexTimer (FTM), [537](#)
- FTM\_DRV\_InitCounter
  - FlexTimer Module Counter Driver (FTM\_MC), [563](#)
- FTM\_DRV\_InitInputCapture
  - FlexTimer Input Capture Driver (FTM\_IC), [558](#)
- FTM\_DRV\_InitOutputCompare
  - FlexTimer Output Compare Driver (FTM\_OC), [569](#)
- FTM\_DRV\_InitPwm
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔PWM), [585](#)
- FTM\_DRV\_IsChnDma
  - FlexTimer (FTM), [537](#)
- FTM\_DRV\_IsChnIcrst
  - FlexTimer (FTM), [538](#)
- FTM\_DRV\_IsFaultFlagDetected
  - FlexTimer (FTM), [538](#)
- FTM\_DRV\_IsFaultInputEnabled
  - FlexTimer (FTM), [538](#)
- FTM\_DRV\_IsFtmEnable
  - FlexTimer (FTM), [539](#)
- FTM\_DRV\_IsWriteProtectionEnabled
  - FlexTimer (FTM), [539](#)
- FTM\_DRV\_MaskOutputChannels
  - FlexTimer (FTM), [539](#)
- FTM\_DRV\_QuadDecodeStart
  - FlexTimer Quadrature Decoder Driver (FTM\_QD), [592](#)
- FTM\_DRV\_QuadDecodeStop

- FlexTimer Quadrature Decoder Driver (FTM\_QD), [592](#)
- FTM\_DRV\_QuadGetState
  - FlexTimer Quadrature Decoder Driver (FTM\_QD), [592](#)
- FTM\_DRV\_SetAllChnSoftwareOutputControl
  - FlexTimer (FTM), [539](#)
- FTM\_DRV\_SetCaptureTestCmd
  - FlexTimer (FTM), [540](#)
- FTM\_DRV\_SetChnDmaCmd
  - FlexTimer (FTM), [540](#)
- FTM\_DRV\_SetChnIcrstCmd
  - FlexTimer (FTM), [541](#)
- FTM\_DRV\_SetChnOutputInitStateCmd
  - FlexTimer (FTM), [541](#)
- FTM\_DRV\_SetChnOutputMask
  - FlexTimer (FTM), [541](#)
- FTM\_DRV\_SetChnSoftwareCtrlCmd
  - FlexTimer (FTM), [541](#)
- FTM\_DRV\_SetChnSoftwareCtrlVal
  - FlexTimer (FTM), [542](#)
- FTM\_DRV\_SetClockFilterPs
  - FlexTimer (FTM), [542](#)
- FTM\_DRV\_SetCountReinitSyncCmd
  - FlexTimer (FTM), [542](#)
- FTM\_DRV\_SetDualChnInvertCmd
  - FlexTimer (FTM), [543](#)
- FTM\_DRV\_SetExtPairDeadtimeValue
  - FlexTimer (FTM), [543](#)
- FTM\_DRV\_SetGlobalLoadCmd
  - FlexTimer (FTM), [543](#)
- FTM\_DRV\_SetGlobalTimeBaseCmd
  - FlexTimer (FTM), [543](#)
- FTM\_DRV\_SetGlobalTimeBaseOutputCmd
  - FlexTimer (FTM), [543](#)
- FTM\_DRV\_SetHalfCycleCmd
  - FlexTimer (FTM), [545](#)
- FTM\_DRV\_SetHalfCycleReloadPoint
  - FlexTimer (FTM), [545](#)
- FTM\_DRV\_SetInitTrigOnReloadCmd
  - FlexTimer (FTM), [545](#)
- FTM\_DRV\_SetInitialCounterValue
  - FlexTimer (FTM), [545](#)
- FTM\_DRV\_SetInvertingControl
  - FlexTimer (FTM), [547](#)
- FTM\_DRV\_SetLoadCmd
  - FlexTimer (FTM), [547](#)
- FTM\_DRV\_SetLoadFreq
  - FlexTimer (FTM), [547](#)
- FTM\_DRV\_SetModuloCounterValue
  - FlexTimer (FTM), [547](#)
- FTM\_DRV\_SetOutputlevel
  - FlexTimer (FTM), [548](#)
- FTM\_DRV\_SetPairDeadtimeCount
  - FlexTimer (FTM), [548](#)
- FTM\_DRV\_SetPairDeadtimePrescale
  - FlexTimer (FTM), [548](#)
- FTM\_DRV\_SetPwmLoadChnSelCmd
  - FlexTimer (FTM), [549](#)
- FTM\_DRV\_SetPwmLoadCmd
  - FlexTimer (FTM), [549](#)
- FTM\_DRV\_SetSoftOutChnValue
  - FlexTimer (FTM), [549](#)
- FTM\_DRV\_SetSoftwareOutputChannelControl
  - FlexTimer (FTM), [550](#)
- FTM\_DRV\_SetSync
  - FlexTimer (FTM), [550](#)
- FTM\_DRV\_SetTrigModeControlCmd
  - FlexTimer (FTM), [550](#)
- FTM\_DRV\_StartNewSignalMeasurement
  - FlexTimer Input Capture Driver (FTM\_IC), [559](#)
- FTM\_DRV\_UpdateOutputCompareChannel
  - FlexTimer Output Compare Driver (FTM\_OC), [569](#)
- FTM\_DRV\_UpdatePwmChannel
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [585](#)
- FTM\_DRV\_UpdatePwmPeriod
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [586](#)
- FTM\_DUTY\_TO\_TICKS\_SHIFT
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [583](#)
- FTM\_EDGE\_DETECT
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_FALLING\_EDGE
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_FALLING\_EDGE\_PERIOD\_MEASUREMENT
  - FlexTimer Input Capture Driver (FTM\_IC), [558](#)
- FTM\_FAULT\_CONTROL\_AUTO\_ALL
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [583](#)
- FTM\_FAULT\_CONTROL\_DISABLED
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [583](#)
- FTM\_FAULT\_CONTROL\_MAN\_ALL
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [583](#)
- FTM\_FAULT\_CONTROL\_MAN\_EVEN
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [583](#)
- FTM\_FAULT\_FLAG
  - FlexTimer (FTM), [530](#)
- FTM\_FAULT\_INT\_ENABLE
  - FlexTimer (FTM), [529](#)
- FTM\_HIGH\_STATE
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [584](#)
- FTM\_IC\_DRV\_SetChannelMode
  - FlexTimer Input Capture Driver (FTM\_IC), [559](#)
- FTM\_LOW\_STATE
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [584](#)
- FTM\_MAIN\_DUPLICATED
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [584](#)
- FTM\_MAIN\_INVERTED

- FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [584](#)
- FTM\_MAX\_DUTY\_CYCLE
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [583](#)
- FTM\_MC\_DRV\_GetDefaultConfig
  - FlexTimer Module Counter Driver (FTM\_MC), [564](#)
- FTM\_MEASURE\_FALLING\_EDGE\_PERIOD
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_MEASURE\_PULSE\_HIGH
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_MEASURE\_PULSE\_LOW
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_MEASURE\_RISING\_EDGE\_PERIOD
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_MODE\_CEN\_ALIGNED\_PWM
  - FlexTimer (FTM), [529](#)
- FTM\_MODE\_EDGE\_ALIGNED\_PWM
  - FlexTimer (FTM), [529](#)
- FTM\_MODE\_EDGE\_ALIGNED\_PWM\_AND\_INPUT\_↔ CAPTURE
  - FlexTimer (FTM), [529](#)
- FTM\_MODE\_INPUT\_CAPTURE
  - FlexTimer (FTM), [529](#)
- FTM\_MODE\_NOT\_INITIALIZED
  - FlexTimer (FTM), [529](#)
- FTM\_MODE\_OUTPUT\_COMPARE
  - FlexTimer (FTM), [529](#)
- FTM\_MODE\_QUADRATURE\_DECODER
  - FlexTimer (FTM), [529](#)
- FTM\_MODE\_UP\_DOWN\_TIMER
  - FlexTimer (FTM), [529](#)
- FTM\_MODE\_UP\_TIMER
  - FlexTimer (FTM), [529](#)
- FTM\_NO\_MEASUREMENT
  - FlexTimer Input Capture Driver (FTM\_IC), [558](#)
- FTM\_NO\_OPERATION
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_NO\_PIN\_CONTROL
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_PERIOD\_OFF\_MEASUREMENT
  - FlexTimer Input Capture Driver (FTM\_IC), [558](#)
- FTM\_PERIOD\_ON\_MEASUREMENT
  - FlexTimer Input Capture Driver (FTM\_IC), [558](#)
- FTM\_POLARITY\_HIGH
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [583](#)
- FTM\_POLARITY\_LOW
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [583](#)
- FTM\_PWM\_SYNC
  - FlexTimer (FTM), [530](#)
- FTM\_PWM\_UPDATE\_IN\_DUTY\_CYCLE
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [583](#)
- FTM\_PWM\_UPDATE\_IN\_TICKS
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔ PWM), [583](#)
- FTM\_QD\_DRV\_GetDefaultConfig
  - FlexTimer Quadrature Decoder Driver (FTM\_QD), [593](#)
- FTM\_QUAD\_COUNT\_AND\_DIR
  - FlexTimer Quadrature Decoder Driver (FTM\_QD), [592](#)
- FTM\_QUAD\_PHASE\_ENCODE
  - FlexTimer Quadrature Decoder Driver (FTM\_QD), [592](#)
- FTM\_QUAD\_PHASE\_INVERT
  - FlexTimer Quadrature Decoder Driver (FTM\_QD), [592](#)
- FTM\_QUAD\_PHASE\_NORMAL
  - FlexTimer Quadrature Decoder Driver (FTM\_QD), [592](#)
- FTM\_RELATIVE\_VALUE
  - FlexTimer Output Compare Driver (FTM\_OC), [568](#)
- FTM\_RELOAD\_FLAG
  - FlexTimer (FTM), [530](#)
- FTM\_RELOAD\_INT\_ENABLE
  - FlexTimer (FTM), [529](#)
- FTM\_RISING\_EDGE
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_RISING\_EDGE\_PERIOD\_MEASUREMENT
  - FlexTimer Input Capture Driver (FTM\_IC), [558](#)
- FTM\_RMW\_CNT
  - FlexTimer (FTM), [526](#)
- FTM\_RMW\_CNTIN
  - FlexTimer (FTM), [526](#)
- FTM\_RMW\_CONF
  - FlexTimer (FTM), [526](#)
- FTM\_RMW\_CnSCV\_REG
  - FlexTimer (FTM), [525](#)
- FTM\_RMW\_DEADTIME
  - FlexTimer (FTM), [526](#)
- FTM\_RMW\_EXTTRIG\_REG
  - FlexTimer (FTM), [526](#)
- FTM\_RMW\_FILTER
  - FlexTimer (FTM), [526](#)
- FTM\_RMW\_FLTCTRL
  - FlexTimer (FTM), [526](#)
- FTM\_RMW\_FMS
  - FlexTimer (FTM), [526](#)
- FTM\_RMW\_MOD
  - FlexTimer (FTM), [526](#)
- FTM\_RMW\_MODE
  - FlexTimer (FTM), [526](#)
- FTM\_RMW\_PAIR0DEADTIME
  - FlexTimer (FTM), [527](#)
- FTM\_RMW\_PAIR1DEADTIME
  - FlexTimer (FTM), [527](#)
- FTM\_RMW\_PAIR2DEADTIME
  - FlexTimer (FTM), [527](#)
- FTM\_RMW\_PAIR3DEADTIME
  - FlexTimer (FTM), [527](#)
- FTM\_RMW\_POL
  - FlexTimer (FTM), [527](#)
- FTM\_RMW\_QDCTRL



- FlexTimer (FTM), [527](#)
- FTM\_RM\_W\_SC
  - FlexTimer (FTM), [527](#)
- FTM\_RM\_W\_STATUS
  - FlexTimer (FTM), [527](#)
- FTM\_RM\_W\_SYNC
  - FlexTimer (FTM), [527](#)
- FTM\_SET\_ON\_MATCH
  - FlexTimer Output Compare Driver (FTM\_OC), [568](#)
- FTM\_SIGNAL\_MEASUREMENT
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_SYSTEM\_CLOCK
  - FlexTimer (FTM), [530](#)
- FTM\_TIME\_OVER\_FLOW\_FLAG
  - FlexTimer (FTM), [530](#)
- FTM\_TIME\_OVER\_FLOW\_INT\_ENABLE
  - FlexTimer (FTM), [529](#)
- FTM\_TIMESTAMP\_BOTH\_EDGES
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_TIMESTAMP\_FALLING\_EDGE
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_TIMESTAMP\_RISING\_EDGE
  - FlexTimer Input Capture Driver (FTM\_IC), [557](#)
- FTM\_TOGGLE\_ON\_MATCH
  - FlexTimer Output Compare Driver (FTM\_OC), [568](#)
- FTM\_UPDATE\_NOW
  - FlexTimer (FTM), [530](#)
- FTM\_WAIT\_LOADING\_POINTS
  - FlexTimer (FTM), [530](#)
- fallingEdgeInterruptCount
  - lin\_state\_t, [655](#)
- fault\_state\_signal\_ptr
  - lin\_node\_attribute\_t, [758](#)
- faultChannelEnabled
  - ftm\_pwm\_ch\_fault\_param\_t, [578](#)
- faultConfig
  - ftm\_pwm\_param\_t, [582](#)
- faultFilterEnabled
  - ftm\_pwm\_ch\_fault\_param\_t, [578](#)
- faultFilterValue
  - ftm\_pwm\_fault\_param\_t, [578](#)
- faultMode
  - ftm\_pwm\_fault\_param\_t, [578](#)
- fdPadding
  - can\_buff\_config\_t, [349](#)
- fifoSize
  - lpspi\_state\_t, [706](#)
- filterEn
  - ftm\_input\_ch\_param\_t, [556](#)
  - ic\_input\_ch\_param\_t, [615](#)
- filterSampleCount
  - cmp\_comparator\_t, [328](#)
- filterSamplePeriod
  - cmp\_comparator\_t, [328](#)
- filterValue
  - ftm\_input\_ch\_param\_t, [556](#)
  - ic\_input\_ch\_param\_t, [615](#)
- finalValue
  - extension\_ftm\_for\_timer\_t, [998](#)
  - ftm\_timer\_param\_t, [562](#)
- fircConfig
  - scg\_config\_t, [295](#)
- firstEdge
  - ftm\_combined\_ch\_param\_t, [581](#)
- fixedChannel
  - cmp\_trigger\_mode\_t, [331](#)
- fixedPort
  - cmp\_trigger\_mode\_t, [331](#)
- flag\_offset
  - lin\_frame\_t, [760](#)
  - lin\_master\_data\_t, [768](#)
- flag\_size
  - lin\_frame\_t, [760](#)
  - lin\_master\_data\_t, [768](#)
- Flash Memory (Flash), [417](#), [437](#)
  - brownOutCode, [434](#)
  - CLEAR\_FTFx\_FSTAT\_ERROR\_BITS, [421](#)
  - CSE\_KEY\_SIZE\_CODE\_MAX, [421](#)
  - CallBack, [434](#)
  - DFLASH\_IFR\_READRESOURCE\_ADDRESS, [421](#)
  - DFlashBase, [434](#), [435](#)
  - DFlashSize, [435](#)
  - EEE\_COMPLETE\_INTERRUPT\_QUICK\_WRITE, [425](#)
  - EEE\_DISABLE, [425](#)
  - EEE\_ENABLE, [425](#)
  - EEE\_QUICK\_WRITE, [425](#)
  - EEE\_STATUS\_QUERY, [425](#)
  - EEESize, [435](#)
  - EERAMBase, [435](#)
  - FLASH\_CALLBACK\_CS, [421](#)
  - FLASH\_DRV\_CheckSum, [425](#)
  - FLASH\_DRV\_ClearReadColisionFlag, [426](#)
  - FLASH\_DRV\_DisableCmdCompleteInterupt, [426](#)
  - FLASH\_DRV\_DisableReadColisionInterupt, [426](#)
  - FLASH\_DRV\_EnableCmdCompleteInterupt, [426](#)
  - FLASH\_DRV\_EnableReadColisionInterupt, [426](#)
  - FLASH\_DRV\_EraseAllBlock, [427](#)
  - FLASH\_DRV\_EraseResume, [427](#)
  - FLASH\_DRV\_EraseSector, [427](#)
  - FLASH\_DRV\_EraseSuspend, [428](#)
  - FLASH\_DRV\_GetCmdCompleteFlag, [428](#)
  - FLASH\_DRV\_GetDefaultConfig, [429](#)
  - FLASH\_DRV\_GetPFlashProtection, [429](#)
  - FLASH\_DRV\_GetReadColisionFlag, [429](#)
  - FLASH\_DRV\_GetSecurityState, [429](#)
  - FLASH\_DRV\_Init, [430](#)
  - FLASH\_DRV\_Program, [430](#)
  - FLASH\_DRV\_ProgramCheck, [430](#)
  - FLASH\_DRV\_ProgramOnce, [431](#)
  - FLASH\_DRV\_ReadOnce, [431](#)
  - FLASH\_DRV\_SecurityBypass, [432](#)
  - FLASH\_DRV\_SetPFlashProtection, [432](#)
  - FLASH\_DRV\_VerifyAllBlock, [433](#)
  - FLASH\_DRV\_VerifySection, [433](#)

- FLASH\_NOT\_SECURE, [421](#)
- FLASH\_SECURE\_BACKDOOR\_DISABLED, [422](#)
- FLASH\_SECURE\_BACKDOOR\_ENABLED, [422](#)
- FLASH\_SECURITY\_STATE\_KEYEN, [422](#)
- FLASH\_SECURITY\_STATE\_UNSECURED, [422](#)
- FTF<sub>x</sub>\_DPHRASE\_SIZE, [422](#)
- FTF<sub>x</sub>\_ERASE\_ALL\_BLOCK, [422](#)
- FTF<sub>x</sub>\_ERASE\_ALL\_BLOCK\_UNSECURE, [422](#)
- FTF<sub>x</sub>\_ERASE\_BLOCK, [422](#)
- FTF<sub>x</sub>\_ERASE\_SECTOR, [422](#)
- FTF<sub>x</sub>\_FSTAT\_ERROR\_BITS, [422](#)
- FTF<sub>x</sub>\_LONGWORD\_SIZE, [422](#)
- FTF<sub>x</sub>\_PFLASH\_SWAP, [422](#)
- FTF<sub>x</sub>\_PHRASE\_SIZE, [422](#)
- FTF<sub>x</sub>\_PROGRAM\_CHECK, [422](#)
- FTF<sub>x</sub>\_PROGRAM\_LONGWORD, [423](#)
- FTF<sub>x</sub>\_PROGRAM\_ONCE, [423](#)
- FTF<sub>x</sub>\_PROGRAM\_PARTITION, [423](#)
- FTF<sub>x</sub>\_PROGRAM\_PHRASE, [423](#)
- FTF<sub>x</sub>\_PROGRAM\_SECTION, [423](#)
- FTF<sub>x</sub>\_READ\_ONCE, [423](#)
- FTF<sub>x</sub>\_READ\_RESOURCE, [423](#)
- FTF<sub>x</sub>\_RSRC\_CODE\_REG, [423](#)
- FTF<sub>x</sub>\_SECURITY\_BY\_PASS, [423](#)
- FTF<sub>x</sub>\_SET\_EERAM, [423](#)
- FTF<sub>x</sub>\_SWAP\_COMPLETE, [423](#)
- FTF<sub>x</sub>\_SWAP\_READY, [423](#)
- FTF<sub>x</sub>\_SWAP\_REPORT\_STATUS, [423](#)
- FTF<sub>x</sub>\_SWAP\_SET\_IN\_COMPLETE, [423](#)
- FTF<sub>x</sub>\_SWAP\_SET\_IN\_PREPARE, [424](#)
- FTF<sub>x</sub>\_SWAP\_SET\_INDICATOR\_ADDR, [424](#)
- FTF<sub>x</sub>\_SWAP\_UNINIT, [424](#)
- FTF<sub>x</sub>\_SWAP\_UPDATE, [424](#)
- FTF<sub>x</sub>\_SWAP\_UPDATE\_ERASED, [424](#)
- FTF<sub>x</sub>\_VERIFY\_ALL\_BLOCK, [424](#)
- FTF<sub>x</sub>\_VERIFY\_BLOCK, [424](#)
- FTF<sub>x</sub>\_VERIFY\_SECTION, [424](#)
- FTF<sub>x</sub>\_WORD\_SIZE, [424](#)
- flash\_callback\_t, [425](#)
- flash\_flexRam\_function\_control\_code\_t, [425](#)
- GET\_BIT\_0\_7, [424](#)
- GET\_BIT\_16\_23, [424](#)
- GET\_BIT\_24\_31, [424](#)
- GET\_BIT\_8\_15, [424](#)
- NULL\_CALLBACK, [425](#)
- numOfRecordReqMaintain, [435](#)
- PFlashBase, [435](#)
- PFlashSize, [435](#)
- RESUME\_WAIT\_CNT, [425](#)
- SUSPEND\_WAIT\_CNT, [425](#)
- sectorEraseCount, [436](#)
- flash\_callback\_t
  - Flash Memory (Flash), [425](#)
- flash\_eeprom\_status\_t, [421](#)
- flash\_flexRam\_function\_control\_code\_t
  - Flash Memory (Flash), [425](#)
- flash\_ssd\_config\_t, [420](#)
- flash\_user\_config\_t, [420](#)
- FlexCAN Driver, [440](#)
  - FLEXCAN\_DISABLE\_MODE, [452](#)
  - FLEXCAN\_DRV\_AbortTransfer, [454](#)
  - FLEXCAN\_DRV\_ConfigRemoteResponseMb, [454](#)
  - FLEXCAN\_DRV\_ConfigRxFifo, [454](#)
  - FLEXCAN\_DRV\_ConfigRxMb, [455](#)
  - FLEXCAN\_DRV\_ConfigTxMb, [455](#)
  - FLEXCAN\_DRV\_Deinit, [455](#)
  - FLEXCAN\_DRV\_GetBitrate, [455](#)
  - FLEXCAN\_DRV\_GetDefaultConfig, [456](#)
  - FLEXCAN\_DRV\_GetErrorStatus, [456](#)
  - FLEXCAN\_DRV\_GetTransferStatus, [456](#)
  - FLEXCAN\_DRV\_Init, [457](#)
  - FLEXCAN\_DRV\_InstallErrorCallback, [457](#)
  - FLEXCAN\_DRV\_InstallEventCallback, [457](#)
  - FLEXCAN\_DRV\_Receive, [457](#)
  - FLEXCAN\_DRV\_ReceiveBlocking, [458](#)
  - FLEXCAN\_DRV\_RxFifo, [458](#)
  - FLEXCAN\_DRV\_RxFifoBlocking, [458](#)
  - FLEXCAN\_DRV\_Send, [459](#)
  - FLEXCAN\_DRV\_SendBlocking, [459](#)
  - FLEXCAN\_DRV\_SetBitrate, [459](#)
  - FLEXCAN\_DRV\_SetRxFifoGlobalMask, [460](#)
  - FLEXCAN\_DRV\_SetRxIndividualMask, [460](#)
  - FLEXCAN\_DRV\_SetRxMaskType, [460](#)
  - FLEXCAN\_DRV\_SetRxMb14Mask, [460](#)
  - FLEXCAN\_DRV\_SetRxMb15Mask, [461](#)
  - FLEXCAN\_DRV\_SetRxMbGlobalMask, [461](#)
  - FLEXCAN\_EVENT\_ERROR, [452](#)
  - FLEXCAN\_EVENT\_RX\_COMPLETE, [452](#)
  - FLEXCAN\_EVENT\_RXFIFO\_COMPLETE, [452](#)
  - FLEXCAN\_EVENT\_RXFIFO\_OVERFLOW, [452](#)
  - FLEXCAN\_EVENT\_RXFIFO\_WARNING, [452](#)
  - FLEXCAN\_EVENT\_TX\_COMPLETE, [452](#)
  - FLEXCAN\_FREEZE\_MODE, [452](#)
  - FLEXCAN\_LISTEN\_ONLY\_MODE, [452](#)
  - FLEXCAN\_LOOPBACK\_MODE, [452](#)
  - FLEXCAN\_MB\_IDLE, [452](#)
  - FLEXCAN\_MB\_RX\_BUSY, [452](#)
  - FLEXCAN\_MB\_TX\_BUSY, [452](#)
  - FLEXCAN\_MSG\_ID\_EXT, [452](#)
  - FLEXCAN\_MSG\_ID\_STD, [452](#)
  - FLEXCAN\_NORMAL\_MODE, [452](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_104, [453](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_112, [453](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_120, [453](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_128, [453](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_16, [453](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_24, [453](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_32, [453](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_40, [453](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_48, [453](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_56, [453](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_64, [453](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_72, [453](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_8, [453](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_80, [453](#)
  - FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_88, [453](#)



- FLEXCAN\_RX\_FIFO\_ID\_FILTERS\_96, 453
- FLEXCAN\_RX\_FIFO\_ID\_FORMAT\_A, 453
- FLEXCAN\_RX\_FIFO\_ID\_FORMAT\_B, 453
- FLEXCAN\_RX\_FIFO\_ID\_FORMAT\_C, 453
- FLEXCAN\_RX\_FIFO\_ID\_FORMAT\_D, 453
- FLEXCAN\_RX\_MASK\_GLOBAL, 453
- FLEXCAN\_RX\_MASK\_INDIVIDUAL, 453
- FLEXCAN\_RXFIFO\_USING\_INTERRUPTS, 453
- flexcan\_callback\_t, 451
- flexcan\_error\_callback\_t, 451
- flexcan\_event\_type\_t, 451
- flexcan\_mb\_state\_t, 452
- flexcan\_msgbuff\_id\_type\_t, 452
- flexcan\_operation\_modes\_t, 452
- flexcan\_rx\_fifo\_id\_element\_format\_t, 452
- flexcan\_rx\_fifo\_id\_filter\_num\_t, 453
- flexcan\_rx\_mask\_type\_t, 453
- flexcan\_rxfifo\_transfer\_type\_t, 453
- flexcan\_state\_t, 451
- FlexCANState, 448
  - callback, 448
  - callbackParam, 448
  - error\_callback, 448
  - errorCallbackParam, 448
  - mbs, 448
  - transferType, 448
- FlexIO Common Driver, 462
  - FLEXIO\_DRIVER\_TYPE\_DMA, 462
  - FLEXIO\_DRIVER\_TYPE\_INTERRUPTS, 462
  - FLEXIO\_DRIVER\_TYPE\_POLLING, 462
  - FLEXIO\_DRV\_DeinitDevice, 462
  - FLEXIO\_DRV\_InitDevice, 464
  - FLEXIO\_DRV\_Reset, 464
  - flexio\_driver\_type\_t, 462
- FlexIO I2C Driver, 465
  - FLEXIO\_I2C\_DRV\_GenerateNineClock, 469
  - FLEXIO\_I2C\_DRV\_GetBusStatus, 469
  - FLEXIO\_I2C\_DRV\_GetDefaultConfig, 469
  - FLEXIO\_I2C\_DRV\_MasterDeinit, 470
  - FLEXIO\_I2C\_DRV\_MasterGetBaudRate, 470
  - FLEXIO\_I2C\_DRV\_MasterGetStatus, 470
  - FLEXIO\_I2C\_DRV\_MasterInit, 470
  - FLEXIO\_I2C\_DRV\_MasterReceiveData, 471
  - FLEXIO\_I2C\_DRV\_MasterReceiveDataBlocking, 471
  - FLEXIO\_I2C\_DRV\_MasterSendData, 471
  - FLEXIO\_I2C\_DRV\_MasterSendDataBlocking, 472
  - FLEXIO\_I2C\_DRV\_MasterSetBaudRate, 472
  - FLEXIO\_I2C\_DRV\_MasterSetSlaveAddr, 473
  - FLEXIO\_I2C\_DRV\_MasterTransferAbort, 473
  - FLEXIO\_I2C\_DRV\_StatusGenerateNineClock, 473
- FlexIO I2S Driver, 474
  - FLEXIO\_I2S\_DRV\_MasterDeinit, 480
  - FLEXIO\_I2S\_DRV\_MasterGetBaudRate, 480
  - FLEXIO\_I2S\_DRV\_MasterGetDefaultConfig, 481
  - FLEXIO\_I2S\_DRV\_MasterGetStatus, 481
  - FLEXIO\_I2S\_DRV\_MasterInit, 481
  - FLEXIO\_I2S\_DRV\_MasterReceiveData, 482
  - FLEXIO\_I2S\_DRV\_MasterReceiveDataBlocking, 482
  - FLEXIO\_I2S\_DRV\_MasterSendData, 482
  - FLEXIO\_I2S\_DRV\_MasterSendDataBlocking, 483
  - FLEXIO\_I2S\_DRV\_MasterSetConfig, 483
  - FLEXIO\_I2S\_DRV\_MasterSetRxBuffer, 483
  - FLEXIO\_I2S\_DRV\_MasterSetTxBuffer, 484
  - FLEXIO\_I2S\_DRV\_MasterTransferAbort, 484
  - FLEXIO\_I2S\_DRV\_SlaveDeinit, 484
  - FLEXIO\_I2S\_DRV\_SlaveGetDefaultConfig, 485
  - FLEXIO\_I2S\_DRV\_SlaveGetStatus, 485
  - FLEXIO\_I2S\_DRV\_SlaveInit, 485
  - FLEXIO\_I2S\_DRV\_SlaveReceiveData, 486
  - FLEXIO\_I2S\_DRV\_SlaveReceiveDataBlocking, 486
  - FLEXIO\_I2S\_DRV\_SlaveSendData, 487
  - FLEXIO\_I2S\_DRV\_SlaveSendDataBlocking, 487
  - FLEXIO\_I2S\_DRV\_SlaveSetConfig, 488
  - FLEXIO\_I2S\_DRV\_SlaveSetRxBuffer, 488
  - FLEXIO\_I2S\_DRV\_SlaveSetTxBuffer, 489
  - FLEXIO\_I2S\_DRV\_SlaveTransferAbort, 489
  - flexio\_i2s\_slave\_state\_t, 480
- FlexIO SPI Driver, 492
  - FLEXIO\_SPI\_DRV\_MasterDeinit, 499
  - FLEXIO\_SPI\_DRV\_MasterGetBaudRate, 499
  - FLEXIO\_SPI\_DRV\_MasterGetDefaultConfig, 500
  - FLEXIO\_SPI\_DRV\_MasterGetStatus, 500
  - FLEXIO\_SPI\_DRV\_MasterInit, 500
  - FLEXIO\_SPI\_DRV\_MasterSetBaudRate, 500
  - FLEXIO\_SPI\_DRV\_MasterTransfer, 501
  - FLEXIO\_SPI\_DRV\_MasterTransferAbort, 501
  - FLEXIO\_SPI\_DRV\_MasterTransferBlocking, 501
  - FLEXIO\_SPI\_DRV\_SlaveDeinit, 502
  - FLEXIO\_SPI\_DRV\_SlaveGetDefaultConfig, 502
  - FLEXIO\_SPI\_DRV\_SlaveGetStatus, 502
  - FLEXIO\_SPI\_DRV\_SlaveInit, 503
  - FLEXIO\_SPI\_DRV\_SlaveTransfer, 503
  - FLEXIO\_SPI\_DRV\_SlaveTransferAbort, 504
  - FLEXIO\_SPI\_DRV\_SlaveTransferBlocking, 504
  - FLEXIO\_SPI\_TRANSFER\_1BYTE, 499
  - FLEXIO\_SPI\_TRANSFER\_2BYTE, 499
  - FLEXIO\_SPI\_TRANSFER\_4BYTE, 499
  - FLEXIO\_SPI\_TRANSFER\_LSB\_FIRST, 499
  - FLEXIO\_SPI\_TRANSFER\_MSB\_FIRST, 499
  - flexio\_spi\_slave\_state\_t, 498
  - flexio\_spi\_transfer\_bit\_order\_t, 499
  - flexio\_spi\_transfer\_size\_t, 499
- FlexIO UART Driver, 506
  - FLEXIO\_UART\_DIRECTION\_RX, 510
  - FLEXIO\_UART\_DIRECTION\_TX, 510
  - FLEXIO\_UART\_DRV\_Deinit, 510
  - FLEXIO\_UART\_DRV\_GetBaudRate, 510
  - FLEXIO\_UART\_DRV\_GetDefaultConfig, 510
  - FLEXIO\_UART\_DRV\_GetStatus, 511
  - FLEXIO\_UART\_DRV\_Init, 511
  - FLEXIO\_UART\_DRV\_ReceiveData, 511
  - FLEXIO\_UART\_DRV\_ReceiveDataBlocking, 512

- FLEXIO\_UART\_DRV\_SendData, [512](#)
- FLEXIO\_UART\_DRV\_SendDataBlocking, [512](#)
- FLEXIO\_UART\_DRV\_SetConfig, [513](#)
- FLEXIO\_UART\_DRV\_SetRxBuffer, [513](#)
- FLEXIO\_UART\_DRV\_SetTxBuffer, [513](#)
- FLEXIO\_UART\_DRV\_TransferAbort, [514](#)
- flexio\_uart\_driver\_direction\_t, [510](#)
- FlexTimer (FTM), [515](#)
  - CHAN0\_IDX, [525](#)
  - CHAN1\_IDX, [525](#)
  - CHAN2\_IDX, [525](#)
  - CHAN3\_IDX, [525](#)
  - CHAN4\_IDX, [525](#)
  - CHAN5\_IDX, [525](#)
  - CHAN6\_IDX, [525](#)
  - CHAN7\_IDX, [525](#)
  - FTM\_BDM\_MODE\_00, [528](#)
  - FTM\_BDM\_MODE\_01, [528](#)
  - FTM\_BDM\_MODE\_10, [528](#)
  - FTM\_BDM\_MODE\_11, [528](#)
  - FTM\_CHANNEL0\_FLAG, [530](#)
  - FTM\_CHANNEL0\_INT\_ENABLE, [529](#)
  - FTM\_CHANNEL1\_FLAG, [530](#)
  - FTM\_CHANNEL1\_INT\_ENABLE, [529](#)
  - FTM\_CHANNEL2\_FLAG, [530](#)
  - FTM\_CHANNEL2\_INT\_ENABLE, [529](#)
  - FTM\_CHANNEL3\_FLAG, [530](#)
  - FTM\_CHANNEL3\_INT\_ENABLE, [529](#)
  - FTM\_CHANNEL4\_FLAG, [530](#)
  - FTM\_CHANNEL4\_INT\_ENABLE, [529](#)
  - FTM\_CHANNEL5\_FLAG, [530](#)
  - FTM\_CHANNEL5\_INT\_ENABLE, [529](#)
  - FTM\_CHANNEL6\_FLAG, [530](#)
  - FTM\_CHANNEL6\_INT\_ENABLE, [529](#)
  - FTM\_CHANNEL7\_FLAG, [530](#)
  - FTM\_CHANNEL7\_INT\_ENABLE, [529](#)
  - FTM\_CHANNEL\_TRIGGER\_FLAG, [530](#)
  - FTM\_CLOCK\_DIVID\_BY\_1, [528](#)
  - FTM\_CLOCK\_DIVID\_BY\_128, [528](#)
  - FTM\_CLOCK\_DIVID\_BY\_16, [528](#)
  - FTM\_CLOCK\_DIVID\_BY\_2, [528](#)
  - FTM\_CLOCK\_DIVID\_BY\_32, [528](#)
  - FTM\_CLOCK\_DIVID\_BY\_4, [528](#)
  - FTM\_CLOCK\_DIVID\_BY\_64, [528](#)
  - FTM\_CLOCK\_DIVID\_BY\_8, [528](#)
  - FTM\_CLOCK\_SOURCE\_EXTERNALCLK, [528](#)
  - FTM\_CLOCK\_SOURCE\_FIXEDCLK, [528](#)
  - FTM\_CLOCK\_SOURCE\_NONE, [528](#)
  - FTM\_CLOCK\_SOURCE\_SYSTEMCLK, [528](#)
  - FTM\_DEADTIME\_DIVID\_BY\_1, [529](#)
  - FTM\_DEADTIME\_DIVID\_BY\_16, [529](#)
  - FTM\_DEADTIME\_DIVID\_BY\_4, [529](#)
  - FTM\_DRV\_ClearChSC, [531](#)
  - FTM\_DRV\_ClearChnEventStatus, [530](#)
  - FTM\_DRV\_ClearFaultFlagDetected, [531](#)
  - FTM\_DRV\_ClearStatusFlags, [531](#)
  - FTM\_DRV\_ConvertFreqToPeriodTicks, [531](#)
  - FTM\_DRV\_CounterReset, [531](#)
  - FTM\_DRV\_Deinit, [532](#)
  - FTM\_DRV\_DisableFaultInt, [532](#)
  - FTM\_DRV\_DisableInterrupts, [532](#)
  - FTM\_DRV\_EnableInterrupts, [532](#)
  - FTM\_DRV\_GenerateHardwareTrigger, [533](#)
  - FTM\_DRV\_GetChInputState, [533](#)
  - FTM\_DRV\_GetChOutputValue, [534](#)
  - FTM\_DRV\_GetChnCountVal, [533](#)
  - FTM\_DRV\_GetChnEdgeLevel, [533](#)
  - FTM\_DRV\_GetChnEventStatus, [534](#)
  - FTM\_DRV\_GetClockFilterPs, [534](#)
  - FTM\_DRV\_GetCounter, [535](#)
  - FTM\_DRV\_GetCounterInitVal, [535](#)
  - FTM\_DRV\_GetDefaultConfig, [535](#)
  - FTM\_DRV\_GetEnabledInterrupts, [535](#)
  - FTM\_DRV\_GetEventStatus, [536](#)
  - FTM\_DRV\_GetFrequency, [536](#)
  - FTM\_DRV\_GetMod, [536](#)
  - FTM\_DRV\_GetStatusFlags, [536](#)
  - FTM\_DRV\_GetTriggerControlled, [537](#)
  - FTM\_DRV\_Init, [537](#)
  - FTM\_DRV\_IsChnDma, [537](#)
  - FTM\_DRV\_IsChnIcrst, [538](#)
  - FTM\_DRV\_IsFaultFlagDetected, [538](#)
  - FTM\_DRV\_IsFaultInputEnabled, [538](#)
  - FTM\_DRV\_IsFtmEnable, [539](#)
  - FTM\_DRV\_IsWriteProtectionEnabled, [539](#)
  - FTM\_DRV\_MaskOutputChannels, [539](#)
  - FTM\_DRV\_SetAllChnSoftwareOutputControl, [539](#)
  - FTM\_DRV\_SetCaptureTestCmd, [540](#)
  - FTM\_DRV\_SetChnDmaCmd, [540](#)
  - FTM\_DRV\_SetChnIcrstCmd, [541](#)
  - FTM\_DRV\_SetChnOutputInitStateCmd, [541](#)
  - FTM\_DRV\_SetChnOutputMask, [541](#)
  - FTM\_DRV\_SetChnSoftwareCtrlCmd, [541](#)
  - FTM\_DRV\_SetChnSoftwareCtrlVal, [542](#)
  - FTM\_DRV\_SetClockFilterPs, [542](#)
  - FTM\_DRV\_SetCountReinitSyncCmd, [542](#)
  - FTM\_DRV\_SetDualChnInvertCmd, [543](#)
  - FTM\_DRV\_SetExtPairDeadtimeValue, [543](#)
  - FTM\_DRV\_SetGlobalLoadCmd, [543](#)
  - FTM\_DRV\_SetGlobalTimeBaseCmd, [543](#)
  - FTM\_DRV\_SetGlobalTimeBaseOutputCmd, [543](#)
  - FTM\_DRV\_SetHalfCycleCmd, [545](#)
  - FTM\_DRV\_SetHalfCycleReloadPoint, [545](#)
  - FTM\_DRV\_SetInitTrigOnReloadCmd, [545](#)
  - FTM\_DRV\_SetInitialCounterValue, [545](#)
  - FTM\_DRV\_SetInvertingControl, [547](#)
  - FTM\_DRV\_SetLoadCmd, [547](#)
  - FTM\_DRV\_SetLoadFreq, [547](#)
  - FTM\_DRV\_SetModuloCounterValue, [547](#)
  - FTM\_DRV\_SetOutputlevel, [548](#)
  - FTM\_DRV\_SetPairDeadtimeCount, [548](#)
  - FTM\_DRV\_SetPairDeadtimePrescale, [548](#)
  - FTM\_DRV\_SetPwmLoadChnSelCmd, [549](#)
  - FTM\_DRV\_SetPwmLoadCmd, [549](#)
  - FTM\_DRV\_SetSoftOutChnValue, [549](#)

- FTM\_DRV\_SetSoftwareOutputChannelControl, 550
- FTM\_DRV\_SetSync, 550
- FTM\_DRV\_SetTrigModeControlCmd, 550
- FTM\_FAULT\_FLAG, 530
- FTM\_FAULT\_INT\_ENABLE, 529
- FTM\_MODE\_CEN\_ALIGNED\_PWM, 529
- FTM\_MODE\_EDGE\_ALIGNED\_PWM, 529
- FTM\_MODE\_EDGE\_ALIGNED\_PWM\_AND\_INCAPTURE\_CAPTURE, 529
- FTM\_MODE\_INPUT\_CAPTURE, 529
- FTM\_MODE\_NOT\_INITIALIZED, 529
- FTM\_MODE\_OUTPUT\_COMPARE, 529
- FTM\_MODE\_QUADRATURE\_DECODER, 529
- FTM\_MODE\_UP\_DOWN\_TIMER, 529
- FTM\_MODE\_UP\_TIMER, 529
- FTM\_PWM\_SYNC, 530
- FTM\_RELOAD\_FLAG, 530
- FTM\_RELOAD\_INT\_ENABLE, 529
- FTM\_RMW\_CNT, 526
- FTM\_RMW\_CNTIN, 526
- FTM\_RMW\_CONF, 526
- FTM\_RMW\_CnSCV\_REG, 525
- FTM\_RMW\_DEADTIME, 526
- FTM\_RMW\_EXTTTRIG\_REG, 526
- FTM\_RMW\_FILTER, 526
- FTM\_RMW\_FLTCTRL, 526
- FTM\_RMW\_FMS, 526
- FTM\_RMW\_MOD, 526
- FTM\_RMW\_MODE, 526
- FTM\_RMW\_PAIR0DEADTIME, 527
- FTM\_RMW\_PAIR1DEADTIME, 527
- FTM\_RMW\_PAIR2DEADTIME, 527
- FTM\_RMW\_PAIR3DEADTIME, 527
- FTM\_RMW\_POL, 527
- FTM\_RMW\_QDCTRL, 527
- FTM\_RMW\_SC, 527
- FTM\_RMW\_STATUS, 527
- FTM\_RMW\_SYNC, 527
- FTM\_SYSTEM\_CLOCK, 530
- FTM\_TIME\_OVER\_FLOW\_FLAG, 530
- FTM\_TIME\_OVER\_FLOW\_INT\_ENABLE, 529
- FTM\_UPDATE\_NOW, 530
- FTM\_WAIT\_LOADING\_POINTS, 530
- ftm\_bdm\_mode\_t, 528
- ftm\_clock\_ps\_t, 528
- ftm\_clock\_source\_t, 528
- ftm\_config\_mode\_t, 528
- ftm\_deadtime\_ps\_t, 529
- ftm\_interrupt\_option\_t, 529
- ftm\_pwm\_sync\_mode\_t, 529
- ftm\_reg\_update\_t, 530
- ftm\_status\_flag\_t, 530
- ftmStatePtr, 552
- g\_ftmBase, 552
- g\_ftmFaultIrqlId, 552
- g\_ftmIrqlId, 552
- g\_ftmOverflowIrqlId, 552
- g\_ftmReloadIrqlId, 552
- FlexTimer Input Capture Driver (FTM\_IC), 553
  - FTM\_BOTH\_EDGES, 557
  - FTM\_DISABLE\_OPERATION, 557
  - FTM\_DRV\_DeinitInputCapture, 558
  - FTM\_DRV\_GetInputCaptureMeasurement, 558
  - FTM\_DRV\_InitInputCapture, 558
  - FTM\_DRV\_StartNewSignalMeasurement, 559
  - FTM\_EDGE\_DETECT, 557
  - FTM\_FALLING\_EDGE, 557
  - FTM\_FALLING\_EDGE\_PERIOD\_MEASUREMENT, 558
  - FTM\_IC\_DRV\_SetChannelMode, 559
  - FTM\_MEASURE\_FALLING\_EDGE\_PERIOD, 557
  - FTM\_MEASURE\_PULSE\_HIGH, 557
  - FTM\_MEASURE\_PULSE\_LOW, 557
  - FTM\_MEASURE\_RISING\_EDGE\_PERIOD, 557
  - FTM\_NO\_MEASUREMENT, 558
  - FTM\_NO\_OPERATION, 557
  - FTM\_NO\_PIN\_CONTROL, 557
  - FTM\_PERIOD\_OFF\_MEASUREMENT, 558
  - FTM\_PERIOD\_ON\_MEASUREMENT, 558
  - FTM\_RISING\_EDGE, 557
  - FTM\_RISING\_EDGE\_PERIOD\_MEASUREMENT, 558
  - FTM\_SIGNAL\_MEASUREMENT, 557
  - FTM\_TIMESTAMP\_BOTH\_EDGES, 557
  - FTM\_TIMESTAMP\_FALLING\_EDGE, 557
  - FTM\_TIMESTAMP\_RISING\_EDGE, 557
  - ftm\_edge\_alignment\_mode\_t, 557
  - ftm\_ic\_op\_mode\_t, 557
  - ftm\_input\_op\_mode\_t, 557
  - ftm\_signal\_measurement\_mode\_t, 557
- FlexTimer Module Counter Driver (FTM\_MC), 561
  - FTM\_DRV\_CounterRead, 563
  - FTM\_DRV\_CounterStart, 563
  - FTM\_DRV\_CounterStop, 563
  - FTM\_DRV\_InitCounter, 563
  - FTM\_MC\_DRV\_GetDefaultConfig, 564
- FlexTimer Output Compare Driver (FTM\_OC), 565
  - FTM\_ABSOLUTE\_VALUE, 568
  - FTM\_CLEAR\_ON\_MATCH, 568
  - FTM\_DISABLE\_OUTPUT, 568
  - FTM\_DRV\_DeinitOutputCompare, 568
  - FTM\_DRV\_InitOutputCompare, 569
  - FTM\_DRV\_UpdateOutputCompareChannel, 569
  - FTM\_RELATIVE\_VALUE, 568
  - FTM\_SET\_ON\_MATCH, 568
  - FTM\_TOGGLE\_ON\_MATCH, 568
  - ftm\_output\_compare\_mode\_t, 568
  - ftm\_output\_compare\_update\_t, 568
- FlexTimer Pulse Width Modulation Driver (FTM\_PWM), 571
  - FTM\_DRV\_ControlChannelOutput, 584
  - FTM\_DRV\_DeinitPwm, 584
  - FTM\_DRV\_FastUpdatePwmChannels, 584
  - FTM\_DRV\_InitPwm, 585
  - FTM\_DRV\_UpdatePwmChannel, 585

- FTM\_DRV\_UpdatePwmPeriod, 586
- FTM\_DUTY\_TO\_TICKS\_SHIFT, 583
- FTM\_FAULT\_CONTROL\_AUTO\_ALL, 583
- FTM\_FAULT\_CONTROL\_DISABLED, 583
- FTM\_FAULT\_CONTROL\_MAN\_ALL, 583
- FTM\_FAULT\_CONTROL\_MAN\_EVEN, 583
- FTM\_HIGH\_STATE, 584
- FTM\_LOW\_STATE, 584
- FTM\_MAIN\_DUPLICATED, 584
- FTM\_MAIN\_INVERTED, 584
- FTM\_MAX\_DUTY\_CYCLE, 583
- FTM\_POLARITY\_HIGH, 583
- FTM\_POLARITY\_LOW, 583
- FTM\_PWM\_UPDATE\_IN\_DUTY\_CYCLE, 583
- FTM\_PWM\_UPDATE\_IN\_TICKS, 583
- ftm\_fault\_mode\_t, 583
- ftm\_polarity\_t, 583
- ftm\_pwm\_update\_option\_t, 583
- ftm\_safe\_state\_polarity\_t, 583
- ftm\_second\_channel\_polarity\_t, 584
- FlexTimer Quadrature Decoder Driver (FTM\_QD), 588
  - FTM\_DRV\_QuadDecodeStart, 592
  - FTM\_DRV\_QuadDecodeStop, 592
  - FTM\_DRV\_QuadGetState, 592
  - FTM\_QD\_DRV\_GetDefaultConfig, 593
  - FTM\_QUAD\_COUNT\_AND\_DIR, 592
  - FTM\_QUAD\_PHASE\_ENCODE, 592
  - FTM\_QUAD\_PHASE\_INVERT, 592
  - FTM\_QUAD\_PHASE\_NORMAL, 592
  - ftm\_quad\_decode\_mode\_t, 591
  - ftm\_quad\_phase\_polarity\_t, 592
- flexcan\_callback\_t
  - FlexCAN Driver, 451
- flexcan\_data\_info\_t, 449
  - data\_length, 449
  - is\_remote, 449
  - msg\_id\_type, 449
- flexcan\_error\_callback\_t
  - FlexCAN Driver, 451
- flexcan\_event\_type\_t
  - FlexCAN Driver, 451
- flexcan\_id\_table\_t, 449
  - id, 449
  - isExtendedFrame, 449
  - isRemoteFrame, 449
- flexcan\_mb\_handle\_t, 447
  - isBlocking, 447
  - isRemote, 447
  - mb\_message, 447
  - mbSema, 447
  - state, 448
- flexcan\_mb\_state\_t
  - FlexCAN Driver, 452
- flexcan\_msgbuff\_id\_type\_t
  - FlexCAN Driver, 452
- flexcan\_msgbuff\_t, 446
  - cs, 447
  - data, 447
  - dataLen, 447
  - msgId, 447
- flexcan\_operation\_modes\_t
  - FlexCAN Driver, 452
- flexcan\_rx\_fifo\_id\_element\_format\_t
  - FlexCAN Driver, 452
- flexcan\_rx\_fifo\_id\_filter\_num\_t
  - FlexCAN Driver, 453
- flexcan\_rx\_mask\_type\_t
  - FlexCAN Driver, 453
- flexcan\_rxfifo\_transfer\_type\_t
  - FlexCAN Driver, 453
- flexcan\_state\_t
  - FlexCAN Driver, 451
- flexcan\_time\_segment\_t, 450
  - phaseSeg1, 450
  - phaseSeg2, 450
  - preDivider, 450
  - propSeg, 450
  - rJumpwidth, 450
- flexcan\_user\_config\_t, 450
  - bitrate, 451
  - flexcanMode, 451
  - is\_rx\_fifo\_needed, 451
  - max\_num\_mb, 451
  - num\_id\_filters, 451
  - transfer\_type, 451
- flexcanMode
  - flexcan\_user\_config\_t, 451
- Flexible I/O (FlexIO), 594
- flexio\_driver\_type\_t
  - FlexIO Common Driver, 462
- flexio\_i2c\_master\_state\_t, 469
- flexio\_i2c\_master\_user\_config\_t, 468
  - baudRate, 468
  - callback, 468
  - callbackParam, 468
  - driverType, 468
  - rxDMACHannel, 468
  - sclPin, 468
  - sdaPin, 468
  - slaveAddress, 469
  - txDMACHannel, 469
- flexio\_i2s\_master\_state\_t, 480
- flexio\_i2s\_master\_user\_config\_t, 477
  - baudRate, 478
  - bitsWidth, 478
  - callback, 478
  - callbackParam, 478
  - driverType, 478
  - rxDMACHannel, 478
  - rxPin, 478
  - sckPin, 478
  - txDMACHannel, 478
  - txPin, 478
  - wsPin, 478
- flexio\_i2s\_slave\_state\_t
  - FlexIO I2S Driver, 480

- flexio\_i2s\_slave\_user\_config\_t, 479
  - bitsWidth, 479
  - callback, 479
  - callbackParam, 479
  - driverType, 479
  - rxDMACHannel, 479
  - rxPin, 479
  - sckPin, 479
  - txDMACHannel, 480
  - txPin, 480
  - wsPin, 480
- flexio\_spi\_master\_state\_t, 498
- flexio\_spi\_master\_user\_config\_t, 495
  - baudRate, 495
  - bitOrder, 495
  - callback, 495
  - callbackParam, 496
  - clockPhase, 496
  - clockPolarity, 496
  - driverType, 496
  - misoPin, 496
  - mosiPin, 496
  - rxDMACHannel, 496
  - sckPin, 496
  - ssPin, 496
  - transferSize, 496
  - txDMACHannel, 496
- flexio\_spi\_slave\_state\_t
  - FlexIO SPI Driver, 498
- flexio\_spi\_slave\_user\_config\_t, 497
  - bitOrder, 497
  - callback, 497
  - callbackParam, 497
  - clockPhase, 497
  - clockPolarity, 497
  - driverType, 497
  - misoPin, 498
  - mosiPin, 498
  - rxDMACHannel, 498
  - sckPin, 498
  - ssPin, 498
  - transferSize, 498
  - txDMACHannel, 498
- flexio\_spi\_transfer\_bit\_order\_t
  - FlexIO SPI Driver, 499
- flexio\_spi\_transfer\_size\_t
  - FlexIO SPI Driver, 499
- flexio\_uart\_driver\_direction\_t
  - FlexIO UART Driver, 510
- flexio\_uart\_state\_t, 509
- flexio\_uart\_user\_config\_t, 508
  - baudRate, 509
  - bitCount, 509
  - callback, 509
  - callbackParam, 509
  - dataPin, 509
  - direction, 509
  - dmaChannel, 509
  - driverType, 509
- fnmc
  - sbc\_sbc\_t, 1014
- fnms
  - sbc\_wtdog\_status\_t, 1022
- frac
  - peripheral\_clock\_config\_t, 296
- frame
  - sbc\_can\_conf\_t, 1018
- frame\_counter
  - lin\_tl\_descriptor\_t, 763
- frame\_data\_ptr
  - lin\_frame\_t, 760
- frame\_start
  - lin\_protocol\_user\_config\_t, 766
- frame\_tbl\_ptr
  - lin\_protocol\_user\_config\_t, 766
- frame\_timeout\_cnt
  - lin\_protocol\_state\_t, 770
- frameSize
  - sbi\_master\_t, 958
  - sbi\_slave\_t, 959
- FreeRTOS, 595
- freeRun
  - lptmr\_config\_t, 722
- freq
  - scg\_sosc\_config\_t, 289
- frm\_id
  - lin\_schedule\_data\_t, 761
- frm\_len
  - lin\_frame\_t, 760
- frm\_offset
  - lin\_frame\_t, 760
  - lin\_master\_data\_t, 768
- frm\_response
  - lin\_frame\_t, 760
- frm\_size
  - lin\_master\_data\_t, 768
- frm\_type
  - lin\_frame\_t, 760
- fscCfg
  - uja113x\_init\_cfg\_t, 145
- ftm\_bdm\_mode\_t
  - FlexTimer (FTM), 528
- ftm\_clock\_ps\_t
  - FlexTimer (FTM), 528
- ftm\_clock\_source\_t
  - FlexTimer (FTM), 528
- ftm\_combined\_ch\_param\_t, 580
  - deadTime, 580
  - enableExternalTrigger, 580
  - enableExternalTriggerOnNextChn, 580
  - enableModifiedCombine, 580
  - enableSecondChannelOutput, 581
  - firstEdge, 581
  - hwChannelId, 581
  - mainChannelPolarity, 581
  - mainChannelSafeState, 581

- secondChannelPolarity, 581
- secondChannelSafeState, 581
- secondEdge, 581
- ftm\_config\_mode\_t
  - FlexTimer (FTM), 528
- ftm\_deadtime\_ps\_t
  - FlexTimer (FTM), 529
- ftm\_edge\_alignment\_mode\_t
  - FlexTimer Input Capture Driver (FTM\_IC), 557
- ftm\_fault\_mode\_t
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔PWM), 583
- ftm\_ic\_op\_mode\_t
  - FlexTimer Input Capture Driver (FTM\_IC), 557
- ftm\_independent\_ch\_param\_t, 579
  - deadTime, 579
  - enableExternalTrigger, 579
  - enableSecondChannelOutput, 579
  - hwChannelId, 579
  - polarity, 579
  - safeState, 580
  - secondChannelPolarity, 580
  - uDutyCyclePercent, 580
- ftm\_input\_ch\_param\_t, 555
  - channelsCallbacks, 555
  - channelsCallbacksParams, 555
  - continuousModeEn, 556
  - edgeAlignement, 556
  - filterEn, 556
  - filterValue, 556
  - hwChannelId, 556
  - inputMode, 556
  - measurementType, 556
- ftm\_input\_op\_mode\_t
  - FlexTimer Input Capture Driver (FTM\_IC), 557
- ftm\_input\_param\_t, 556
  - inputChConfig, 556
  - nMaxCountValue, 557
  - nNumChannels, 557
- ftm\_interrupt\_option\_t
  - FlexTimer (FTM), 529
- ftm\_output\_cmp\_ch\_param\_t, 567
  - chMode, 567
  - comparedValue, 567
  - enableExternalTrigger, 567
  - hwChannelId, 567
- ftm\_output\_cmp\_param\_t, 567
  - maxCountValue, 567
  - mode, 567
  - nNumOutputChannels, 568
  - outputChannelConfig, 568
- ftm\_output\_compare\_mode\_t
  - FlexTimer Output Compare Driver (FTM\_OC), 568
- ftm\_output\_compare\_update\_t
  - FlexTimer Output Compare Driver (FTM\_OC), 568
- ftm\_phase\_params\_t, 590
  - phaseFilterVal, 590
  - phaseInputFilter, 590
  - phasePolarity, 590
- ftm\_polarity\_t
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔PWM), 583
- ftm\_pwm\_ch\_fault\_param\_t, 578
  - faultChannelEnabled, 578
  - faultFilterEnabled, 578
  - ftmFaultPinPolarity, 578
- ftm\_pwm\_fault\_param\_t, 578
  - faultFilterValue, 578
  - faultMode, 578
  - ftmFaultChannelParam, 578
  - pwmFaultInterrupt, 579
  - pwmOutputStateOnFault, 579
- ftm\_pwm\_param\_t, 581
  - deadTimePrescaler, 582
  - deadTimeValue, 582
  - faultConfig, 582
  - mode, 582
  - nNumCombinedPwmChannels, 582
  - nNumIndependentPwmChannels, 582
  - pwmCombinedChannelConfig, 582
  - pwmIndependentChannelConfig, 582
  - uFrequencyHZ, 582
- ftm\_pwm\_sync\_mode\_t
  - FlexTimer (FTM), 529
- ftm\_pwm\_sync\_t, 522
  - autoClearTrigger, 523
  - hardwareSync0, 523
  - hardwareSync1, 523
  - hardwareSync2, 523
  - initCounterSync, 523
  - inverterSync, 523
  - maskRegSync, 523
  - maxLoadingPoint, 523
  - minLoadingPoint, 523
  - outRegSync, 524
  - softwareSync, 524
  - syncPoint, 524
- ftm\_pwm\_update\_option\_t
  - FlexTimer Pulse Width Modulation Driver (FTM\_↔PWM), 583
- ftm\_quad\_decode\_config\_t, 590
  - initialVal, 590
  - maxVal, 590
  - mode, 590
  - phaseAConfig, 591
  - phaseBConfig, 591
- ftm\_quad\_decode\_mode\_t
  - FlexTimer Quadrature Decoder Driver (FTM\_QD), 591
- ftm\_quad\_decoder\_state\_t, 591
  - counter, 591
  - counterDirection, 591
  - overflowDirection, 591
  - overflowFlag, 591
- ftm\_quad\_phase\_polarity\_t



- FlexTimer Quadrature Decoder Driver (FTM\_QD), 592
- ftm\_reg\_update\_t
  - FlexTimer (FTM), 530
- ftm\_safe\_state\_polarity\_t
  - FlexTimer Pulse Width Modulation Driver (FTM\_PWM), 583
- ftm\_second\_channel\_polarity\_t
  - FlexTimer Pulse Width Modulation Driver (FTM\_PWM), 584
- ftm\_signal\_measurement\_mode\_t
  - FlexTimer Input Capture Driver (FTM\_IC), 557
- ftm\_state\_t, 521
  - channelsCallbacks, 522
  - channelsCallbacksParams, 522
  - enableNotification, 522
  - ftmClockSource, 522
  - ftmModValue, 522
  - ftmMode, 522
  - ftmPeriod, 522
  - ftmSourceClockFrequency, 522
  - measurementResults, 522
- ftm\_status\_flag\_t
  - FlexTimer (FTM), 530
- ftm\_timer\_param\_t, 562
  - finalValue, 562
  - initialValue, 562
  - mode, 563
- ftm\_user\_config\_t, 524
  - BDMMode, 524
  - enableInitializationTrigger, 524
  - ftmClockSource, 524
  - ftmMode, 524
  - ftmPrescaler, 524
  - isTofIsrEnabled, 525
  - syncMethod, 525
- ftmClockSource
  - extension\_ftm\_for\_ic\_t, 617
  - extension\_ftm\_for\_oc\_t, 839
  - ftm\_state\_t, 522
  - ftm\_user\_config\_t, 524
- ftmFaultChannelParam
  - ftm\_pwm\_fault\_param\_t, 578
- ftmFaultPinPolarity
  - ftm\_pwm\_ch\_fault\_param\_t, 578
- ftmModValue
  - ftm\_state\_t, 522
- ftmMode
  - ftm\_state\_t, 522
  - ftm\_user\_config\_t, 524
- ftmPeriod
  - ftm\_state\_t, 522
- ftmPrescaler
  - extension\_ftm\_for\_ic\_t, 617
  - extension\_ftm\_for\_oc\_t, 839
  - ftm\_user\_config\_t, 524
- ftmSourceClockFrequency
  - ftm\_state\_t, 522
- ftmStatePtr
  - FlexTimer (FTM), 552
- fullSize
  - csec\_state\_t, 261
- function
  - lin\_protocol\_user\_config\_t, 766
- function\_id
  - lin\_product\_id\_t, 1085
- g\_RtcClkInFreq
  - Clock Manager Driver, 315
- g\_TCikFreq
  - Clock Manager Driver, 315
- g\_buffer\_backup\_data
  - Low level API, 782
- g\_drivers
  - 00. Library internals, 149
- g\_ftmBase
  - FlexTimer (FTM), 552
- g\_ftmFaultIrqId
  - FlexTimer (FTM), 552
- g\_ftmIrqId
  - FlexTimer (FTM), 552
- g\_ftmOverflowIrqId
  - FlexTimer (FTM), 552
- g\_ftmReloadIrqId
  - FlexTimer (FTM), 552
- g\_lin\_flag\_handle\_tbl
  - Low level API, 782
- g\_lin\_frame\_data\_buffer
  - Low level API, 782
- g\_lin\_frame\_flag\_handle\_tbl
  - Low level API, 782
- g\_lin\_frame\_updating\_flag\_tbl
  - Low level API, 782
- g\_lin\_hardware\_ifc
  - Low level API, 782
- g\_lin\_master\_data\_array
  - Low level API, 782
- g\_lin\_node\_attribute\_array
  - Low level API, 782
- g\_lin\_protocol\_state\_array
  - Low level API, 782
- g\_lin\_protocol\_user\_cfg\_array
  - Low level API, 782
- g\_lin\_tl\_descriptor\_array
  - Low level API, 782
- g\_lin\_virtual\_ifc
  - Low level API, 783
- g\_linLpuartIsrs
  - LIN Driver, 666
- g\_lpspiBase
  - LPSPi Driver, 718
- g\_lpspiIrqId
  - LPSPi Driver, 718
- g\_lpspiStatePtr
  - LPSPi Driver, 718
- g\_xtal0ClkFreq
  - Clock Manager Driver, 315

- GENERAL\_REJECT
  - Common Transport Layer API, [318](#)
- GET\_BIT\_0\_7
  - Flash Memory (Flash), [424](#)
- GET\_BIT\_16\_23
  - Flash Memory (Flash), [424](#)
- GET\_BIT\_24\_31
  - Flash Memory (Flash), [424](#)
- GET\_BIT\_8\_15
  - Flash Memory (Flash), [424](#)
- GIS\_ADDRESS
  - Register address map, [921](#)
- GO\_TO\_SLEEP\_SET
  - Common Core API., [316](#)
- GPIO\_INPUT\_DIRECTION
  - PINS Driver, [858](#)
- GPIO\_OUTPUT\_DIRECTION
  - PINS Driver, [858](#)
- GPIO\_UNSPECIFIED\_DIRECTION
  - PINS Driver, [858](#)
- gPowerManagerState
  - Power Manager, [874](#)
- gain
  - scg\_sosc\_config\_t, [290](#)
- gating
  - module\_clk\_config\_t, [298](#)
- glEvt
  - sbc\_evn\_capt\_t, [1027](#)
- go\_to\_sleep\_flg
  - lin\_protocol\_state\_t, [770](#)
  - lin\_word\_status\_str\_t, [756](#)
- gpioBase
  - pin\_settings\_config\_t, [858](#)
- groupConfigArray
  - adc\_config\_t, [236](#)
- groupIndex
  - adc\_callback\_info\_t, [1081](#)
- HHSDC\_IO1HOC\_DISABLED
  - HVIO high-side configuration options (0x71), [597](#)
- HHSDC\_IO1HOC\_ENABLED
  - HVIO high-side configuration options (0x71), [597](#)
- HHSDC\_IO2HOC\_DISABLED
  - HVIO high-side configuration options (0x71), [597](#)
- HHSDC\_IO2HOC\_ENABLED
  - HVIO high-side configuration options (0x71), [597](#)
- HHSDC\_IO3HOC\_DISABLED
  - HVIO high-side configuration options (0x71), [597](#)
- HHSDC\_IO3HOC\_ENABLED
  - HVIO high-side configuration options (0x71), [597](#)
- HHSDC\_IO4HOC\_DISABLED
  - HVIO high-side configuration options (0x71), [597](#)
- HHSDC\_IO4HOC\_ENABLED
  - HVIO high-side configuration options (0x71), [597](#)
- HHSDC\_IO5HOC\_DISABLED
  - HVIO high-side configuration options (0x71), [597](#)
- HHSDC\_IO5HOC\_ENABLED
  - HVIO high-side configuration options (0x71), [598](#)
- HHSDC\_IO6HOC\_DISABLED
  - HVIO high-side configuration options (0x71), [598](#)
- HHSDC\_IO6HOC\_ENABLED
  - HVIO high-side configuration options (0x71), [598](#)
- HHSDC\_IO7HOC\_DISABLED
  - HVIO high-side configuration options (0x71), [598](#)
- HHSDC\_IO7HOC\_ENABLED
  - HVIO high-side configuration options (0x71), [598](#)
- HHSDC\_IO8HOC\_DISABLED
  - HVIO high-side configuration options (0x71), [598](#)
- HHSDC\_IO8HOC\_ENABLED
  - HVIO high-side configuration options (0x71), [598](#)
- HLSDC\_IO1LOC\_DISABLED
  - HVIO low-side configuration options (0x72), [600](#)
- HLSDC\_IO1LOC\_ENABLED
  - HVIO low-side configuration options (0x72), [600](#)
- HLSDC\_IO2LOC\_DISABLED
  - HVIO low-side configuration options (0x72), [600](#)
- HLSDC\_IO2LOC\_ENABLED
  - HVIO low-side configuration options (0x72), [600](#)
- HLSDC\_IO3LOC\_DISABLED
  - HVIO low-side configuration options (0x72), [600](#)
- HLSDC\_IO3LOC\_ENABLED
  - HVIO low-side configuration options (0x72), [600](#)
- HLSDC\_IO4LOC\_DISABLED
  - HVIO low-side configuration options (0x72), [600](#)
- HLSDC\_IO4LOC\_ENABLED
  - HVIO low-side configuration options (0x72), [600](#)
- HLSDC\_IO5LOC\_DISABLED
  - HVIO low-side configuration options (0x72), [600](#)
- HLSDC\_IO5LOC\_ENABLED
  - HVIO low-side configuration options (0x72), [601](#)
- HLSDC\_IO6LOC\_DISABLED
  - HVIO low-side configuration options (0x72), [601](#)
- HLSDC\_IO6LOC\_ENABLED
  - HVIO low-side configuration options (0x72), [601](#)
- HLSDC\_IO7LOC\_DISABLED
  - HVIO low-side configuration options (0x72), [601](#)
- HLSDC\_IO7LOC\_ENABLED
  - HVIO low-side configuration options (0x72), [601](#)
- HLSDC\_IO8LOC\_DISABLED
  - HVIO low-side configuration options (0x72), [601](#)
- HLSDC\_IO8LOC\_ENABLED
  - HVIO low-side configuration options (0x72), [601](#)
- HOURS\_IN\_A\_DAY
  - RTC Driver, [899](#)
- HSRUN\_MODE
  - Clock Manager Driver, [305](#)
- HVIO high-side configuration options (0x71), [596](#)
  - HHSDC\_IO1HOC\_DISABLED, [597](#)
  - HHSDC\_IO1HOC\_ENABLED, [597](#)
  - HHSDC\_IO2HOC\_DISABLED, [597](#)
  - HHSDC\_IO2HOC\_ENABLED, [597](#)
  - HHSDC\_IO3HOC\_DISABLED, [597](#)
  - HHSDC\_IO3HOC\_ENABLED, [597](#)
  - HHSDC\_IO4HOC\_DISABLED, [597](#)
  - HHSDC\_IO4HOC\_ENABLED, [597](#)
  - HHSDC\_IO5HOC\_DISABLED, [597](#)
  - HHSDC\_IO5HOC\_ENABLED, [598](#)



- HHSDC\_IO6HOC\_DISABLED, [598](#)
- HHSDC\_IO6HOC\_ENABLED, [598](#)
- HHSDC\_IO7HOC\_DISABLED, [598](#)
- HHSDC\_IO7HOC\_ENABLED, [598](#)
- HHSDC\_IO8HOC\_DISABLED, [598](#)
- HHSDC\_IO8HOC\_ENABLED, [598](#)
- HVIO low-side configuration options (0x72), [599](#)
  - HLSDC\_IO1LOC\_DISABLED, [600](#)
  - HLSDC\_IO1LOC\_ENABLED, [600](#)
  - HLSDC\_IO2LOC\_DISABLED, [600](#)
  - HLSDC\_IO2LOC\_ENABLED, [600](#)
  - HLSDC\_IO3LOC\_DISABLED, [600](#)
  - HLSDC\_IO3LOC\_ENABLED, [600](#)
  - HLSDC\_IO4LOC\_DISABLED, [600](#)
  - HLSDC\_IO4LOC\_ENABLED, [600](#)
  - HLSDC\_IO5LOC\_DISABLED, [600](#)
  - HLSDC\_IO5LOC\_ENABLED, [601](#)
  - HLSDC\_IO6LOC\_DISABLED, [601](#)
  - HLSDC\_IO6LOC\_ENABLED, [601](#)
  - HLSDC\_IO7LOC\_DISABLED, [601](#)
  - HLSDC\_IO7LOC\_ENABLED, [601](#)
  - HLSDC\_IO8LOC\_DISABLED, [601](#)
  - HLSDC\_IO8LOC\_ENABLED, [601](#)
- HVIO1C\_ADDRESS
  - Register address map, [921](#)
- HVIO2C\_ADDRESS
  - Register address map, [921](#)
- HVIO3C\_ADDRESS
  - Register address map, [922](#)
- HVIO4C\_ADDRESS
  - Register address map, [922](#)
- HVIO5C\_ADDRESS
  - Register address map, [922](#)
- HVIO6C\_ADDRESS
  - Register address map, [922](#)
- HVIO7C\_ADDRESS
  - Register address map, [922](#)
- HVIO8C\_ADDRESS
  - Register address map, [922](#)
- HVIOHSDRVC\_ADDRESS
  - Register address map, [922](#)
- HVIOLSDRVC\_ADDRESS
  - Register address map, [922](#)
- haltOnError
  - edma\_user\_config\_t, [377](#)
- hardwareSync0
  - ftm\_pwm\_sync\_t, [523](#)
- hardwareSync1
  - ftm\_pwm\_sync\_t, [523](#)
- hardwareSync2
  - ftm\_pwm\_sync\_t, [523](#)
- hccrConfig
  - scg\_clock\_mode\_config\_t, [294](#)
- highSideCtrl
  - uja113x\_mtpnv\_cfg\_t, [208](#)
- hour
  - rtc\_timedate\_t, [895](#)
- hvioBank0Cfg
  - uja113x\_init\_cfg\_t, [145](#)
- hvioBank1Cfg
  - uja113x\_init\_cfg\_t, [145](#)
- hvioConfig
  - uja113x\_hvio\_bank\_cfg\_t, [144](#)
- hwAverage
  - adc\_average\_config\_t, [218](#)
- hwAvgEnable
  - adc\_average\_config\_t, [218](#)
- hwChannelId
  - ftm\_combined\_ch\_param\_t, [581](#)
  - ftm\_independent\_ch\_param\_t, [579](#)
  - ftm\_input\_ch\_param\_t, [556](#)
  - ftm\_output\_cmp\_ch\_param\_t, [567](#)
  - ic\_input\_ch\_param\_t, [615](#)
  - oc\_output\_ch\_param\_t, [838](#)
- hwTriggerSupport
  - adc\_group\_config\_t, [235](#)
- hysteresisLevel
  - cmp\_comparator\_t, [328](#)
- I2C\_GetDefaultMasterConfig
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [627](#)
- I2C\_GetDefaultSlaveConfig
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [627](#)
- I2C\_MasterAbortTransfer
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [628](#)
- I2C\_MasterDeinit
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [628](#)
- I2C\_MasterGetBaudRate
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [628](#)
- I2C\_MasterGetTransferStatus
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [628](#)
- I2C\_MasterInit
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [629](#)
- I2C\_MasterReceiveData
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [629](#)
- I2C\_MasterReceiveDataBlocking
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [629](#)
- I2C\_MasterSendData
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [630](#)
- I2C\_MasterSendDataBlocking
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [630](#)
- I2C\_MasterSetBaudRate
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [630](#)
- I2C\_MasterSetSlaveAddress

- Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [631](#)
- I2C\_PAL\_FAST\_MODE
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [627](#)
- I2C\_PAL\_FASTPLUS\_MODE
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [627](#)
- I2C\_PAL\_HIGHSPEED\_MODE
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [627](#)
- I2C\_PAL\_STANDARD\_MODE
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [627](#)
- I2C\_PAL\_ULTRAFAST\_MODE
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [627](#)
- I2C\_PAL\_USING\_DMA
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [627](#)
- I2C\_PAL\_USING\_INTERRUPTS
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [627](#)
- I2C\_SlaveAbortTransfer
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [631](#)
- I2C\_SlaveDeinit
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [631](#)
- I2C\_SlaveGetTransferStatus
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [631](#)
- I2C\_SlaveInit
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [631](#)
- I2C\_SlaveReceiveData
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [632](#)
- I2C\_SlaveReceiveDataBlocking
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [632](#)
- I2C\_SlaveSendData
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [632](#)
- I2C\_SlaveSendDataBlocking
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [633](#)
- I2C\_SlaveSetRxBuffer
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [633](#)
- I2C\_SlaveSetTxBuffer
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [633](#)
- I2S - Peripheral Abstraction Layer (I2S PAL), [602](#)
  - I2S\_Abort, [606](#)
  - I2S\_Deinit, [607](#)
  - I2S\_GetBaudRate, [607](#)
  - I2S\_GetDefaultConfig, [607](#)
  - I2S\_GetStatus, [607](#)
  - I2S\_Init, [608](#)
  - I2S\_MASTER, [606](#)
  - I2S\_ReceiveData, [608](#)
  - I2S\_ReceiveDataBlocking, [608](#)
  - I2S\_SLAVE, [606](#)
  - I2S\_SendData, [608](#)
  - I2S\_SendDataBlocking, [609](#)
  - I2S\_SetRxBuffer, [609](#)
  - I2S\_SetTxBuffer, [609](#)
  - I2S\_USING\_DMA, [606](#)
  - I2S\_USING\_INTERRUPT, [606](#)
  - i2s\_mode\_t, [606](#)
  - i2s\_transfer\_type\_t, [606](#)
- I2S\_Abort
  - I2S - Peripheral Abstraction Layer (I2S PAL), [606](#)
- I2S\_Deinit
  - I2S - Peripheral Abstraction Layer (I2S PAL), [607](#)
- I2S\_GetBaudRate
  - I2S - Peripheral Abstraction Layer (I2S PAL), [607](#)
- I2S\_GetDefaultConfig
  - I2S - Peripheral Abstraction Layer (I2S PAL), [607](#)
- I2S\_GetStatus
  - I2S - Peripheral Abstraction Layer (I2S PAL), [607](#)
- I2S\_Init
  - I2S - Peripheral Abstraction Layer (I2S PAL), [608](#)
- I2S\_MASTER
  - I2S - Peripheral Abstraction Layer (I2S PAL), [606](#)
- I2S\_ReceiveData
  - I2S - Peripheral Abstraction Layer (I2S PAL), [608](#)
- I2S\_ReceiveDataBlocking
  - I2S - Peripheral Abstraction Layer (I2S PAL), [608](#)
- I2S\_SLAVE
  - I2S - Peripheral Abstraction Layer (I2S PAL), [606](#)
- I2S\_SendData
  - I2S - Peripheral Abstraction Layer (I2S PAL), [608](#)
- I2S\_SendDataBlocking
  - I2S - Peripheral Abstraction Layer (I2S PAL), [609](#)
- I2S\_SetRxBuffer
  - I2S - Peripheral Abstraction Layer (I2S PAL), [609](#)
- I2S\_SetTxBuffer
  - I2S - Peripheral Abstraction Layer (I2S PAL), [609](#)
- I2S\_USING\_DMA
  - I2S - Peripheral Abstraction Layer (I2S PAL), [606](#)
- I2S\_USING\_INTERRUPT
  - I2S - Peripheral Abstraction Layer (I2S PAL), [606](#)
- i2c\_instance\_t, [1083](#)
  - instIdx, [1083](#)
  - instType, [1083](#)
- i2c\_master\_t, [624](#)
  - baudRate, [625](#)
  - callback, [625](#)
  - callbackParam, [625](#)
  - dmaChannel1, [625](#)
  - dmaChannel2, [625](#)
  - extension, [625](#)
  - is10bitAddr, [625](#)
  - operatingMode, [625](#)

- slaveAddress, [625](#)
  - transferType, [625](#)
- i2c\_operating\_mode\_t
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [627](#)
- i2c\_pal\_transfer\_type\_t
  - Inter Integrated Circuit - Peripheral Abstraction Layer(I2C PAL), [627](#)
- i2c\_slave\_t, [626](#)
  - callback, [626](#)
  - callbackParam, [626](#)
  - dmaChannel, [626](#)
  - is10bitAddr, [626](#)
  - operatingMode, [626](#)
  - slaveAddress, [626](#)
  - slaveListening, [626](#)
  - transferType, [626](#)
- i2s\_instance\_t, [1084](#)
  - instIdx, [1084](#)
  - instType, [1084](#)
- i2s\_mode\_t
  - I2S - Peripheral Abstraction Layer (I2S PAL), [606](#)
- i2s\_transfer\_type\_t
  - I2S - Peripheral Abstraction Layer (I2S PAL), [606](#)
- i2s\_user\_config\_t, [604](#)
  - baudRate, [604](#)
  - callback, [604](#)
  - callbackParam, [605](#)
  - extension, [605](#)
  - mode, [605](#)
  - rxDMACHannel, [605](#)
  - transferType, [605](#)
  - txDMACHannel, [605](#)
  - wordWidth, [605](#)
- IC\_DISABLE\_OPERATION
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [617](#)
- IC\_Deinit
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [617](#)
- IC\_DisableNotification
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [618](#)
- IC\_EnableNotification
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [618](#)
- IC\_GetMeasurement
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [618](#)
- IC\_Init
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [618](#)
- IC\_MEASURE\_FALLING\_EDGE\_PERIOD
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [617](#)
- IC\_MEASURE\_PULSE\_HIGH
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [617](#)
- IC\_MEASURE\_PULSE\_LOW
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [617](#)
- IC\_MEASURE\_RISING\_EDGE\_PERIOD
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [617](#)
- IC\_SetChannelMode
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [619](#)
- IC\_StartChannel
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [619](#)
- IC\_StopChannel
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [619](#)
- IC\_TIMESTAMP\_BOTH\_EDGES
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [617](#)
- IC\_TIMESTAMP\_FALLING\_EDGE
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [617](#)
- IC\_TIMESTAMP\_RISING\_EDGE
  - Input Capture - Peripheral Abstraction Layer (I↔C PAL), [617](#)
- ID0\_ADDRESS
  - Register address map, [922](#)
- ID1\_ADDRESS
  - Register address map, [922](#)
- ID2\_ADDRESS
  - Register address map, [922](#)
- ID3\_ADDRESS
  - Register address map, [923](#)
- IDM0\_ADDRESS
  - Register address map, [923](#)
- IDM1\_ADDRESS
  - Register address map, [923](#)
- IDM2\_ADDRESS
  - Register address map, [923](#)
- IDM3\_ADDRESS
  - Register address map, [923](#)
- INT\_SYS\_DisableIRQ
  - Interrupt Manager (Interrupt), [638](#)
- INT\_SYS\_DisableIRQGlobal
  - Interrupt Manager (Interrupt), [638](#)
- INT\_SYS\_EnableIRQ
  - Interrupt Manager (Interrupt), [639](#)
- INT\_SYS\_EnableIRQGlobal
  - Interrupt Manager (Interrupt), [639](#)
- INT\_SYS\_GetPriority
  - Interrupt Manager (Interrupt), [639](#)
- INT\_SYS\_InstallHandler
  - Interrupt Manager (Interrupt), [639](#)
- INT\_SYS\_SetPriority
  - Interrupt Manager (Interrupt), [639](#)
- IO15FEIE\_DISABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [248](#)
- IO15FEIE\_ENABLE

- Bank N wake-up interrupt enable configuration options (0x37/0x47), [248](#)
- IO15OLIE\_DISABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [245](#)
- IO15OLIE\_ENABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [245](#)
- IO15REIE\_DISABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [248](#)
- IO15REIE\_ENABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [248](#)
- IO15SCIE\_DISABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [245](#)
- IO15SCIE\_ENABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [245](#)
- IO26FEIE\_DISABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [248](#)
- IO26FEIE\_ENABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [248](#)
- IO26OLIE\_DISABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [245](#)
- IO26OLIE\_ENABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [245](#)
- IO26REIE\_DISABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [248](#)
- IO26REIE\_ENABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [249](#)
- IO26SCIE\_DISABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [245](#)
- IO26SCIE\_ENABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [246](#)
- IO2SFC\_LIMP\_HOME
  - Start-up control configuration options (0x73), [971](#)
- IO2SFC\_STD\_IO
  - Start-up control configuration options (0x73), [971](#)
- IO37FEIE\_DISABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [249](#)
- IO37FEIE\_ENABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [249](#)
- IO37OLIE\_DISABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [246](#)
- IO37OLIE\_ENABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [246](#)
- IO37REIE\_DISABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [249](#)
- IO37REIE\_ENABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [249](#)
- IO37SCIE\_DISABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [246](#)
- IO37SCIE\_ENABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [246](#)
- IO3SFC\_LIMP\_HOME
  - Start-up control configuration options (0x73), [971](#)
- IO3SFC\_STD\_IO
  - Start-up control configuration options (0x73), [971](#)
- IO48FEIE\_DISABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [249](#)
- IO48FEIE\_ENABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [249](#)
- IO48OLIE\_DISABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [246](#)
- IO48OLIE\_ENABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [246](#)
- IO48REIE\_DISABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [249](#)
- IO48REIE\_ENABLE
  - Bank N wake-up interrupt enable configuration options (0x37/0x47), [249](#)
- IO48SCIE\_DISABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [246](#)
- IO48SCIE\_ENABLE
  - Bank N fail interrupt enable configuration options (0x38/0x48), [246](#)
- IO4SFC\_LIMP\_HOME
  - Start-up control configuration options (0x73), [971](#)
- IO4SFC\_STD\_IO
  - Start-up control configuration options (0x73), [971](#)
- IONAC\_DEACTIVATED
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONAC\_ENABLED
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONAC\_HVIO
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONAC\_HVIO\_INV
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONAC\_TMR\_1
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONAC\_TMR\_2
  - 04. HVIO bank 0 & 1 control, [190](#)

- IONAC\_TMR\_3
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONAC\_TMR\_4
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONCC\_HS
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONCC\_HS\_SLOPE
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONCC\_HS\_WUP
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONCC\_HS\_WUP\_SLOPE
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONCC\_LS
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONCC\_LS\_SLOPE
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONCC\_OFF
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONCC\_WUP
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONOLTC\_10\_MA
  - 02. Battery monitor configuration, [168](#)
- IONOLTC\_20\_MA
  - 02. Battery monitor configuration, [168](#)
- IONOLTC\_2\_MA
  - 02. Battery monitor configuration, [167](#)
- IONOLTC\_5\_MA
  - 02. Battery monitor configuration, [167](#)
- IONSC\_FULL
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONSC\_NONE
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONSC\_OVERVOLTAGE
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONSC\_UNDRVOLTAGE
  - 04. HVIO bank 0 & 1 control, [190](#)
- IONSCTC\_30\_MA
  - 02. Battery monitor configuration, [168](#)
- IONSCTC\_45\_MA
  - 02. Battery monitor configuration, [168](#)
- IONSCTC\_60\_MA
  - 02. Battery monitor configuration, [168](#)
- IONSCTC\_90\_MA
  - 02. Battery monitor configuration, [168](#)
- ic\_config\_t, [615](#)
  - extension, [616](#)
  - inputChConfig, [616](#)
  - nNumChannels, [616](#)
- ic\_input\_ch\_param\_t, [615](#)
  - channelCallbackParams, [615](#)
  - channelCallbacks, [615](#)
  - channelExtension, [615](#)
  - filterEn, [615](#)
  - filterValue, [615](#)
  - hwChannelId, [615](#)
  - inputCaptureMode, [615](#)
- ic\_instance\_t, [1084](#)
  - instIdx, [1085](#)
  - instType, [1085](#)
- ic\_option\_mode\_t
  - Input Capture - Peripheral Abstraction Layer (IC PAL), [617](#)
- ic\_pal\_state\_t, [617](#)
- id
  - can\_message\_t, [349](#)
  - flexcan\_id\_table\_t, [449](#)
  - uja113x\_cpnc\_cfg\_t, [152](#)
- idFilterTable
  - extension\_flexcan\_rx\_fifo\_t, [351](#)
- idFormat
  - extension\_flexcan\_rx\_fifo\_t, [351](#)
- idMask
  - uja113x\_cpnc\_cfg\_t, [152](#)
- idType
  - can\_buff\_config\_t, [349](#)
- ide
  - sbc\_frame\_t, [1018](#)
  - uja113x\_cpnc\_cfg\_t, [152](#)
- identif
  - sbc\_can\_conf\_t, [1019](#)
- idle\_timeout\_cnt
  - lin\_protocol\_state\_t, [770](#)
- inOutMappingConfig
  - trgmux\_user\_config\_t, [987](#)
- index
  - csec\_state\_t, [261](#)
- init
  - crc\_spec\_t, [208](#)
- initCounterSync
  - ftm\_pwm\_sync\_t, [523](#)
- initValue
  - pin\_settings\_config\_t, [858](#)
- initial\_NAD
  - lin\_node\_attribute\_t, [758](#)
- initialVal
  - ftm\_quad\_decode\_config\_t, [590](#)
- initialValue
  - ftm\_timer\_param\_t, [562](#)
- Initialization, [610](#)
  - ld\_init, [610](#)
- initialize
  - pmc\_lpo\_clock\_config\_t, [297](#)
  - scg\_clock\_mode\_config\_t, [294](#)
  - scg\_clockout\_config\_t, [294](#)
  - scg\_firc\_config\_t, [292](#)
  - scg\_rtc\_config\_t, [293](#)
  - scg\_sirc\_config\_t, [291](#)
  - scg\_sosc\_config\_t, [290](#)
  - scg\_spill\_config\_t, [293](#)
  - sim\_clock\_out\_config\_t, [284](#)
  - sim\_lpo\_clock\_config\_t, [285](#)
  - sim\_plat\_gate\_config\_t, [286](#)
  - sim\_tclk\_config\_t, [285](#)
  - sim\_trace\_clock\_config\_t, [287](#)
- Input Capture - Peripheral Abstraction Layer (IC PAL), [611](#)

- IC\_DISABLE\_OPERATION, 617
- IC\_Deinit, 617
- IC\_DisableNotification, 618
- IC\_EnableNotification, 618
- IC\_GetMeasurement, 618
- IC\_Init, 618
- IC\_MEASURE\_FALLING\_EDGE\_PERIOD, 617
- IC\_MEASURE\_PULSE\_HIGH, 617
- IC\_MEASURE\_PULSE\_LOW, 617
- IC\_MEASURE\_RISING\_EDGE\_PERIOD, 617
- IC\_SetChannelMode, 619
- IC\_StartChannel, 619
- IC\_StopChannel, 619
- IC\_TIMESTAMP\_BOTH\_EDGES, 617
- IC\_TIMESTAMP\_FALLING\_EDGE, 617
- IC\_TIMESTAMP\_RISING\_EDGE, 617
- ic\_option\_mode\_t, 617
- inputBuff
  - csec\_state\_t, 261
- inputCaptureMode
  - ic\_input\_ch\_param\_t, 615
- inputChConfig
  - ftm\_input\_param\_t, 556
  - ic\_config\_t, 616
- inputChannelArray
  - adc\_group\_config\_t, 235
- inputClock
  - adc\_converter\_config\_t, 217
  - extension\_adc\_s32k1xx\_t, 236
- inputMode
  - ftm\_input\_ch\_param\_t, 556
- insertDeadtime
  - pwm\_channel\_t, 888
- instIdx
  - adc\_instance\_t, 1082
  - can\_instance\_t, 1082
  - i2c\_instance\_t, 1083
  - i2s\_instance\_t, 1084
  - ic\_instance\_t, 1085
  - mpu\_instance\_t, 1086
  - oc\_instance\_t, 1087
  - pwm\_instance\_t, 1087
  - spi\_instance\_t, 1088
  - timing\_instance\_t, 1089
  - uart\_instance\_t, 1090
  - wdg\_instance\_t, 1090
- instType
  - adc\_instance\_t, 1082
  - can\_instance\_t, 1082
  - i2c\_instance\_t, 1083
  - i2s\_instance\_t, 1084
  - ic\_instance\_t, 1085
  - mpu\_instance\_t, 1086
  - oc\_instance\_t, 1087
  - pwm\_instance\_t, 1088
  - spi\_instance\_t, 1088
  - timing\_instance\_t, 1089
  - uart\_instance\_t, 1090
  - wdg\_instance\_t, 1090
- intEnable
  - pdb\_timer\_config\_t, 849
  - wdg\_config\_t, 1063
  - wdog\_user\_config\_t, 1071
- Inter Integrated Circuit - Peripheral Abstraction Layer(↔ I2C PAL), 620
  - I2C\_GetDefaultMasterConfig, 627
  - I2C\_GetDefaultSlaveConfig, 627
  - I2C\_MasterAbortTransfer, 628
  - I2C\_MasterDeinit, 628
  - I2C\_MasterGetBaudRate, 628
  - I2C\_MasterGetTransferStatus, 628
  - I2C\_MasterInit, 629
  - I2C\_MasterReceiveData, 629
  - I2C\_MasterReceiveDataBlocking, 629
  - I2C\_MasterSendData, 630
  - I2C\_MasterSendDataBlocking, 630
  - I2C\_MasterSetBaudRate, 630
  - I2C\_MasterSetSlaveAddress, 631
  - I2C\_PAL\_FAST\_MODE, 627
  - I2C\_PAL\_FASTPLUS\_MODE, 627
  - I2C\_PAL\_HIGHSPEED\_MODE, 627
  - I2C\_PAL\_STANDARD\_MODE, 627
  - I2C\_PAL\_ULTRAFast\_MODE, 627
  - I2C\_PAL\_USING\_DMA, 627
  - I2C\_PAL\_USING\_INTERRUPTS, 627
  - I2C\_SlaveAbortTransfer, 631
  - I2C\_SlaveDeinit, 631
  - I2C\_SlaveGetTransferStatus, 631
  - I2C\_SlaveInit, 631
  - I2C\_SlaveReceiveData, 632
  - I2C\_SlaveReceiveDataBlocking, 632
  - I2C\_SlaveSendData, 632
  - I2C\_SlaveSendDataBlocking, 633
  - I2C\_SlaveSetRxBuffer, 633
  - I2C\_SlaveSetTxBuffer, 633
  - i2c\_operating\_mode\_t, 627
  - i2c\_pal\_transfer\_type\_t, 627
- Interface management, 635
  - l\_ifc\_goto\_sleep, 635
  - l\_ifc\_init, 635
  - l\_ifc\_read\_status, 636
  - l\_ifc\_wake\_up, 636
- interleave\_timeout\_counter
  - lin\_tl\_descriptor\_t, 764
- Interrupt Manager (Interrupt), 637
  - DefaultISR, 638
  - INT\_SYS\_DisableIRQ, 638
  - INT\_SYS\_DisableIRQGlobal, 638
  - INT\_SYS\_EnableIRQ, 639
  - INT\_SYS\_EnableIRQGlobal, 639
  - INT\_SYS\_GetPriority, 639
  - INT\_SYS\_InstallHandler, 639
  - INT\_SYS\_SetPriority, 639
  - isr\_t, 638
- Interrupt vector numbers for S32K142, 642
- interruptCf



- erm\_user\_config\_t, [403](#)
- interruptEnable
  - adc\_chan\_config\_t, [219](#)
  - edma\_transfer\_config\_t, [381](#)
  - ewm\_init\_config\_t, [408](#)
  - lptmr\_config\_t, [723](#)
- inverterState
  - cmp\_comparator\_t, [328](#)
- inverterSync
  - ftm\_pwm\_sync\_t, [523](#)
- io15OpenLoadThrCtrl
  - uja113x\_boldt\_cfg\_t, [189](#)
- io15ShortCircuitThrCtrl
  - uja113x\_bscdt\_cfg\_t, [188](#)
- io26OpenLoadThrCtrl
  - uja113x\_boldt\_cfg\_t, [189](#)
- io26ShortCircuitThrCtrl
  - uja113x\_bscdt\_cfg\_t, [188](#)
- io37OpenLoadThrCtrl
  - uja113x\_boldt\_cfg\_t, [189](#)
- io37ShortCircuitThrCtrl
  - uja113x\_bscdt\_cfg\_t, [188](#)
- io48OpenLoadThrCtrl
  - uja113x\_boldt\_cfg\_t, [189](#)
- io48ShortCircuitThrCtrl
  - uja113x\_bscdt\_cfg\_t, [188](#)
- ionActivateCtrl
  - uja113x\_hvio\_cfg\_t, [188](#)
- ionConfCtrl
  - uja113x\_hvio\_cfg\_t, [188](#)
- ionShutdownCtrl
  - uja113x\_hvio\_cfg\_t, [188](#)
- is10bitAddr
  - i2c\_master\_t, [625](#)
  - i2c\_slave\_t, [626](#)
  - lpi2c\_master\_user\_config\_t, [674](#)
  - lpi2c\_slave\_user\_config\_t, [675](#)
- is\_remote
  - flexcan\_data\_info\_t, [449](#)
- is\_rx\_fifo\_needed
  - flexcan\_user\_config\_t, [451](#)
- isBlocking
  - flexcan\_mb\_handle\_t, [447](#)
  - lpspi\_state\_t, [706](#)
- isBusBusy
  - lin\_state\_t, [655](#)
- isExtendedFrame
  - flexcan\_id\_table\_t, [449](#)
- isInit
  - drv\_config\_t, [1083](#)
- isInterruptEnabled
  - lpit\_user\_channel\_config\_t, [691](#)
- isPcsContinuous
  - lpspi\_master\_config\_t, [704](#)
  - lpspi\_state\_t, [706](#)
- isRemote
  - can\_buff\_config\_t, [349](#)
  - flexcan\_mb\_handle\_t, [447](#)
- isRemoteFrame
  - flexcan\_id\_table\_t, [449](#)
- isRxBlocking
  - lin\_state\_t, [655](#)
  - lpuart\_state\_t, [733](#)
- isRxBusy
  - lin\_state\_t, [655](#)
  - lpuart\_state\_t, [733](#)
- isToflsrEnabled
  - ftm\_user\_config\_t, [525](#)
- isTransferInProgress
  - lpspi\_state\_t, [706](#)
- isTxBlocking
  - lin\_state\_t, [656](#)
  - lpuart\_state\_t, [733](#)
- isTxBusy
  - lin\_state\_t, [656](#)
  - lpuart\_state\_t, [733](#)
- isr\_t
  - Interrupt Manager (Interrupt), [638](#)
- iv
  - csec\_state\_t, [261](#)
- J2602 Specific API, [643](#)
- J2602 Transport Layer specific API, [644](#)
- keyId
  - csec\_state\_t, [261](#)
- L1WIE\_DISABLE
  - Transceiver interrupt enable configuration options (0x23), [1004](#)
- L1WIE\_ENABLE
  - Transceiver interrupt enable configuration options (0x23), [1004](#)
- L2WIE\_DISABLE
  - Transceiver interrupt enable configuration options (0x23), [1004](#)
- L2WIE\_ENABLE
  - Transceiver interrupt enable configuration options (0x23), [1004](#)
- l\_diagnostic\_mode\_t
  - Low level API, [774](#)
- l\_ifc\_goto\_sleep
  - Interface management, [635](#)
- l\_ifc\_init
  - Interface management, [635](#)
- l\_ifc\_read\_status
  - Interface management, [636](#)
- l\_ifc\_wake\_up
  - Interface management, [636](#)
- l\_sch\_set
  - Schedule management, [932](#)
- l\_sch\_tick
  - Schedule management, [932](#)
- l\_sys\_init
  - Driver and cluster management, [370](#)
- l\_sys\_irq\_disable
  - User provided call-outs, [1060](#)

- `l_sys_irq_restore`
  - User provided call-outs, [1060](#)
- `LCKC_ADDRESS`
  - Register address map, [923](#)
- `LD_ANY_FUNCTION`
  - Common Transport Layer API, [319](#)
- `LD_ANY_MESSAGE`
  - Common Transport Layer API, [319](#)
- `LD_ANY_SUPPLIER`
  - Common Transport Layer API, [319](#)
- `LD_BROADCAST`
  - Common Transport Layer API, [319](#)
- `LD_CHECK_N_AS_TIMEOUT`
  - Low level API, [776](#)
- `LD_CHECK_N_CR_TIMEOUT`
  - Low level API, [776](#)
- `LD_COMPLETED`
  - Low level API, [776](#)
- `LD_DATA_AVAILABLE`
  - Low level API, [774](#)
- `LD_DATA_ERROR`
  - Common Transport Layer API, [319](#)
- `LD_DIAG_IDLE`
  - Low level API, [775](#)
- `LD_DIAG_RX_FUNCTIONAL`
  - Low level API, [775](#)
- `LD_DIAG_RX_INTERLEAVED`
  - Low level API, [775](#)
- `LD_DIAG_RX_PHY`
  - Low level API, [775](#)
- `LD_DIAG_TX_FUNCTIONAL`
  - Low level API, [775](#)
- `LD_DIAG_TX_INTERLEAVED`
  - Low level API, [775](#)
- `LD_DIAG_TX_PHY`
  - Low level API, [775](#)
- `LD_FAILED`
  - Low level API, [776](#)
- `LD_FUNCTIONAL_NAD`
  - Common Transport Layer API, [319](#)
- `LD_IN_PROGRESS`
  - Low level API, [776](#)
- `LD_LENGTH_NOT_CORRECT`
  - Common Transport Layer API, [319](#)
- `LD_LENGTH_TOO_SHORT`
  - Common Transport Layer API, [319](#)
- `LD_N_AS_TIMEOUT`
  - Low level API, [776](#)
- `LD_N_CR_TIMEOUT`
  - Low level API, [776](#)
- `LD_NEGATIVE`
  - Low level API, [775](#)
- `LD_NEGATIVE_RESPONSE`
  - Low level API, [771](#)
- `LD_NO_CHECK_TIMEOUT`
  - Low level API, [776](#)
- `LD_NO_DATA`
  - Low level API, [774](#)
- `LD_NO_MSG`
  - Low level API, [776](#)
- `LD_NO_RESPONSE`
  - Low level API, [775](#)
- `LD_OVERWRITTEN`
  - Low level API, [775](#)
- `LD_POSITIVE_RESPONSE`
  - Low level API, [771](#)
- `LD_QUEUE_AVAILABLE`
  - Low level API, [774](#)
- `LD_QUEUE_EMPTY`
  - Low level API, [774](#)
- `LD_QUEUE_FULL`
  - Low level API, [774](#)
- `LD_READ_OK`
  - Common Transport Layer API, [319](#)
- `LD_RECEIVE_ERROR`
  - Low level API, [774](#)
- `LD_REQUEST_FINISHED`
  - Low level API, [777](#)
- `LD_SERVICE_BUSY`
  - Low level API, [777](#)
- `LD_SERVICE_ERROR`
  - Low level API, [777](#)
- `LD_SERVICE_IDLE`
  - Low level API, [777](#)
- `LD_SET_OK`
  - Common Transport Layer API, [319](#)
- `LD_SUCCESS`
  - Low level API, [775](#)
- `LD_TRANSFER_ERROR`
  - Low level API, [774](#)
- `LD_TRANSMIT_ERROR`
  - Low level API, [774](#)
- `LD_WRONG_SN`
  - Low level API, [776](#)
- `LIN 2.1 Specific API`, [645](#)
  - `lin_collision_resolve`, [645](#)
  - `lin_make_res_evnt_frame`, [645](#)
  - `lin_update_err_signal`, [646](#)
  - `lin_update_rx_evnt_frame`, [646](#)
  - `lin_update_word_status_lin21`, [646](#)
- `LIN Core API`, [647](#)
- `LIN Driver`, [648](#)
  - `CHECK_PARITY`, [657](#)
  - `g_linLpuartIsrs`, [666](#)
  - `LIN_BAUDRATE_ADJUSTED`, [657](#)
  - `LIN_CHECKSUM_ERROR`, [657](#)
  - `LIN_DRV_AbortTransferData`, [658](#)
  - `LIN_DRV_AutoBaudCapture`, [658](#)
  - `LIN_DRV_Deinit`, [659](#)
  - `LIN_DRV_DisableIRQ`, [659](#)
  - `LIN_DRV_EnableIRQ`, [659](#)
  - `LIN_DRV_GetCurrentNodeState`, [659](#)
  - `LIN_DRV_GetDefaultConfig`, [660](#)
  - `LIN_DRV_GetReceiveStatus`, [660](#)
  - `LIN_DRV_GetTransmitStatus`, [660](#)
  - `LIN_DRV_GoToSleepMode`, [661](#)



- LIN\_DRV\_GotIdleState, [661](#)
- LIN\_DRV\_IRQHandler, [662](#)
- LIN\_DRV\_Init, [661](#)
- LIN\_DRV\_InstallCallback, [661](#)
- LIN\_DRV\_MakeChecksumByte, [662](#)
- LIN\_DRV\_MasterSendHeader, [662](#)
- LIN\_DRV\_ProcessParity, [663](#)
- LIN\_DRV\_ReceiveFrameData, [663](#)
- LIN\_DRV\_ReceiveFrameDataBlocking, [664](#)
- LIN\_DRV\_SendFrameData, [664](#)
- LIN\_DRV\_SendFrameDataBlocking, [665](#)
- LIN\_DRV\_SendWakeupSignal, [665](#)
- LIN\_DRV\_SetTimeoutCounter, [666](#)
- LIN\_DRV\_TimeoutService, [666](#)
- LIN\_FRAME\_ERROR, [657](#)
- LIN\_NO\_EVENT, [657](#)
- LIN\_NODE\_STATE\_IDLE, [658](#)
- LIN\_NODE\_STATE\_RECV\_DATA, [658](#)
- LIN\_NODE\_STATE\_RECV\_DATA\_COMPLETED, [658](#)
- LIN\_NODE\_STATE\_RECV\_PID, [658](#)
- LIN\_NODE\_STATE\_RECV\_SYNC, [658](#)
- LIN\_NODE\_STATE\_SEND\_BREAK\_FIELD, [658](#)
- LIN\_NODE\_STATE\_SEND\_DATA, [658](#)
- LIN\_NODE\_STATE\_SEND\_DATA\_COMPLETED, [658](#)
- LIN\_NODE\_STATE\_SEND\_PID, [658](#)
- LIN\_NODE\_STATE\_SLEEP\_MODE, [658](#)
- LIN\_NODE\_STATE\_UNINIT, [658](#)
- LIN\_PID\_ERROR, [657](#)
- LIN\_PID\_OK, [657](#)
- LIN\_READBACK\_ERROR, [657](#)
- LIN\_RECV\_BREAK\_FIELD\_OK, [657](#)
- LIN\_RX\_COMPLETED, [658](#)
- LIN\_RX\_OVERRUN, [658](#)
- LIN\_SYNC\_ERROR, [657](#)
- LIN\_SYNC\_OK, [657](#)
- LIN\_TX\_COMPLETED, [657](#)
- LIN\_WAKEUP\_SIGNAL, [657](#)
- lin\_callback\_t, [657](#)
- lin\_event\_id\_t, [657](#)
- lin\_node\_state\_t, [658](#)
- lin\_timer\_get\_time\_interval\_t, [657](#)
- MAKE\_PARITY, [657](#)
- MASTER, [657](#)
- SLAVE, [657](#)
- LIN Stack, [667](#)
- LIN\_BAUDRATE\_ADJUSTED
  - LIN Driver, [657](#)
- LIN\_CHECKSUM\_ERROR
  - LIN Driver, [657](#)
- LIN\_DIAGNOSTIC\_CLASS\_I
  - Low level API, [774](#)
- LIN\_DIAGNOSTIC\_CLASS\_II
  - Low level API, [774](#)
- LIN\_DIAGNOSTIC\_CLASS\_III
  - Low level API, [774](#)
- LIN\_DRV\_AbortTransferData
  - LIN Driver, [658](#)
- LIN\_DRV\_AutoBaudCapture
  - LIN Driver, [658](#)
- LIN\_DRV\_Deinit
  - LIN Driver, [659](#)
- LIN\_DRV\_DisableIRQ
  - LIN Driver, [659](#)
- LIN\_DRV\_EnableIRQ
  - LIN Driver, [659](#)
- LIN\_DRV\_GetCurrentNodeState
  - LIN Driver, [659](#)
- LIN\_DRV\_GetDefaultConfig
  - LIN Driver, [660](#)
- LIN\_DRV\_GetReceiveStatus
  - LIN Driver, [660](#)
- LIN\_DRV\_GetTransmitStatus
  - LIN Driver, [660](#)
- LIN\_DRV\_GoToSleepMode
  - LIN Driver, [661](#)
- LIN\_DRV\_GotIdleState
  - LIN Driver, [661](#)
- LIN\_DRV\_IRQHandler
  - LIN Driver, [662](#)
- LIN\_DRV\_Init
  - LIN Driver, [661](#)
- LIN\_DRV\_InstallCallback
  - LIN Driver, [661](#)
- LIN\_DRV\_MakeChecksumByte
  - LIN Driver, [662](#)
- LIN\_DRV\_MasterSendHeader
  - LIN Driver, [662](#)
- LIN\_DRV\_ProcessParity
  - LIN Driver, [663](#)
- LIN\_DRV\_ReceiveFrameData
  - LIN Driver, [663](#)
- LIN\_DRV\_ReceiveFrameDataBlocking
  - LIN Driver, [664](#)
- LIN\_DRV\_SendFrameData
  - LIN Driver, [664](#)
- LIN\_DRV\_SendFrameDataBlocking
  - LIN Driver, [665](#)
- LIN\_DRV\_SendWakeupSignal
  - LIN Driver, [665](#)
- LIN\_DRV\_SetTimeoutCounter
  - LIN Driver, [666](#)
- LIN\_DRV\_TimeoutService
  - LIN Driver, [666](#)
- LIN\_FRAME\_ERROR
  - LIN Driver, [657](#)
- LIN\_FRM\_DIAG
  - Low level API, [775](#)
- LIN\_FRM\_EVNT
  - Low level API, [775](#)
- LIN\_FRM\_SPRDC
  - Low level API, [775](#)
- LIN\_FRM\_UNCD
  - Low level API, [775](#)
- LIN\_LLD\_BUS\_ACTIVITY\_TIMEOUT

- Low level API, [776](#)
- LIN\_LLD\_CHECKSUM\_ERR
  - Low level API, [776](#)
- LIN\_LLD\_ERROR
  - Low level API, [771](#)
- LIN\_LLD\_FRAME\_ERR
  - Low level API, [776](#)
- LIN\_LLD\_NODATA\_TIMEOUT
  - Low level API, [776](#)
- LIN\_LLD\_OK
  - Low level API, [771](#)
- LIN\_LLD\_PID\_ERR
  - Low level API, [776](#)
- LIN\_LLD\_PID\_OK
  - Low level API, [776](#)
- LIN\_LLD\_READBACK\_ERR
  - Low level API, [776](#)
- LIN\_LLD\_RX\_COMPLETED
  - Low level API, [776](#)
- LIN\_LLD\_TX\_COMPLETED
  - Low level API, [776](#)
- LIN\_MASTER
  - Low level API, [771](#)
- LIN\_NO\_EVENT
  - LIN Driver, [657](#)
- LIN\_NODE\_STATE\_IDLE
  - LIN Driver, [658](#)
- LIN\_NODE\_STATE\_RECV\_DATA
  - LIN Driver, [658](#)
- LIN\_NODE\_STATE\_RECV\_DATA\_COMPLETED
  - LIN Driver, [658](#)
- LIN\_NODE\_STATE\_RECV\_PID
  - LIN Driver, [658](#)
- LIN\_NODE\_STATE\_RECV\_SYNC
  - LIN Driver, [658](#)
- LIN\_NODE\_STATE\_SEND\_BREAK\_FIELD
  - LIN Driver, [658](#)
- LIN\_NODE\_STATE\_SEND\_DATA
  - LIN Driver, [658](#)
- LIN\_NODE\_STATE\_SEND\_DATA\_COMPLETED
  - LIN Driver, [658](#)
- LIN\_NODE\_STATE\_SEND\_PID
  - LIN Driver, [658](#)
- LIN\_NODE\_STATE\_SLEEP\_MODE
  - LIN Driver, [658](#)
- LIN\_NODE\_STATE\_UNINIT
  - LIN Driver, [658](#)
- LIN\_PID\_ERROR
  - LIN Driver, [657](#)
- LIN\_PID\_OK
  - LIN Driver, [657](#)
- LIN\_PRODUCT\_ID
  - Common Transport Layer API, [319](#)
- LIN\_PROTOCOL\_13
  - Low level API, [776](#)
- LIN\_PROTOCOL\_20
  - Low level API, [776](#)
- LIN\_PROTOCOL\_21
  - Low level API, [776](#)
- Low level API, [776](#)
- LIN\_PROTOCOL\_J2602
  - Low level API, [776](#)
- LIN\_READ\_USR\_DEF\_MAX
  - Low level API, [771](#)
- LIN\_READ\_USR\_DEF\_MIN
  - Low level API, [771](#)
- LIN\_READBACK\_ERROR
  - LIN Driver, [657](#)
- LIN\_RECV\_BREAK\_FIELD\_OK
  - LIN Driver, [657](#)
- LIN\_RES\_PUB
  - Low level API, [775](#)
- LIN\_RES\_SUB
  - Low level API, [775](#)
- LIN\_RX\_COMPLETED
  - LIN Driver, [658](#)
- LIN\_RX\_OVERRUN
  - LIN Driver, [658](#)
- LIN\_SCH\_TBL\_COLL\_RESOLV
  - Low level API, [777](#)
- LIN\_SCH\_TBL\_DIAG
  - Low level API, [777](#)
- LIN\_SCH\_TBL\_GO\_TO\_SLEEP
  - Low level API, [777](#)
- LIN\_SCH\_TBL\_NORM
  - Low level API, [777](#)
- LIN\_SCH\_TBL\_NULL
  - Low level API, [777](#)
- LIN\_SERIAL\_NUMBER
  - Common Transport Layer API, [320](#)
- LIN\_SLAVE
  - Low level API, [772](#)
- LIN\_SYNC\_ERROR
  - LIN Driver, [657](#)
- LIN\_SYNC\_OK
  - LIN Driver, [657](#)
- LIN\_TL\_CALLBACK\_HANDLER
  - Low level API, [772](#)
- LIN\_TX\_COMPLETED
  - LIN Driver, [657](#)
- LIN\_WAKEUP\_SIGNAL
  - LIN Driver, [657](#)
- LINC\_ADDRESS
  - Register address map, [923](#)
- LK0C
  - UJA116xA SBC Driver, [1033](#)
- LK1C
  - UJA116xA SBC Driver, [1033](#)
- LK2C
  - UJA116xA SBC Driver, [1033](#)
- LK3C
  - UJA116xA SBC Driver, [1033](#)
- LK4C
  - UJA116xA SBC Driver, [1033](#)
- LK5C
  - UJA116xA SBC Driver, [1033](#)
- LK6C

- UJA116xA SBC Driver, [1033](#)
- LKAC
  - UJA116xA SBC Driver, [1033](#)
- LMC1\_TRANSCEIVER\_LISTEN
  - 02. LIN configuration, [170](#)
- LMC1\_TRANSCEIVER\_OFF
  - 02. LIN configuration, [170](#)
- LMC1\_TRANSCEIVER\_ON
  - 02. LIN configuration, [170](#)
- LMC1\_TRANSCEIVER\_ON2
  - 02. LIN configuration, [170](#)
- LMC2\_TRANSCEIVER\_LISTEN
  - 02. LIN configuration, [171](#)
- LMC2\_TRANSCEIVER\_OFF
  - 02. LIN configuration, [171](#)
- LMC2\_TRANSCEIVER\_ON
  - 02. LIN configuration, [171](#)
- LMC2\_TRANSCEIVER\_ON2
  - 02. LIN configuration, [171](#)
- LPI2C Driver, [670](#)
  - LPI2C\_DRV\_MasterAbortTransferData, [676](#)
  - LPI2C\_DRV\_MasterDeinit, [677](#)
  - LPI2C\_DRV\_MasterGetBaudRate, [677](#)
  - LPI2C\_DRV\_MasterGetDefaultConfig, [677](#)
  - LPI2C\_DRV\_MasterGetTransferStatus, [677](#)
  - LPI2C\_DRV\_MasterIRQHandler, [678](#)
  - LPI2C\_DRV\_MasterInit, [678](#)
  - LPI2C\_DRV\_MasterReceiveData, [678](#)
  - LPI2C\_DRV\_MasterReceiveDataBlocking, [679](#)
  - LPI2C\_DRV\_MasterSendData, [679](#)
  - LPI2C\_DRV\_MasterSendDataBlocking, [679](#)
  - LPI2C\_DRV\_MasterSetBaudRate, [680](#)
  - LPI2C\_DRV\_MasterSetSlaveAddr, [680](#)
  - LPI2C\_DRV\_SetMasterBusIdleTimeout, [680](#)
  - LPI2C\_DRV\_SlaveAbortTransferData, [681](#)
  - LPI2C\_DRV\_SlaveDeinit, [681](#)
  - LPI2C\_DRV\_SlaveGetDefaultConfig, [681](#)
  - LPI2C\_DRV\_SlaveGetTransferStatus, [681](#)
  - LPI2C\_DRV\_SlaveIRQHandler, [683](#)
  - LPI2C\_DRV\_SlaveInit, [681](#)
  - LPI2C\_DRV\_SlaveReceiveData, [683](#)
  - LPI2C\_DRV\_SlaveReceiveDataBlocking, [683](#)
  - LPI2C\_DRV\_SlaveSendData, [684](#)
  - LPI2C\_DRV\_SlaveSendDataBlocking, [684](#)
  - LPI2C\_DRV\_SlaveSetRxBuffer, [684](#)
  - LPI2C\_DRV\_SlaveSetTxBuffer, [685](#)
  - LPI2C\_FAST\_MODE, [676](#)
  - LPI2C\_STANDARD\_MODE, [676](#)
  - LPI2C\_USING\_DMA, [676](#)
  - LPI2C\_USING\_INTERRUPTS, [676](#)
  - lpi2c\_mode\_t, [676](#)
  - lpi2c\_transfer\_type\_t, [676](#)
- LPI2C\_DRV\_MasterAbortTransferData
  - LPI2C Driver, [676](#)
- LPI2C\_DRV\_MasterDeinit
  - LPI2C Driver, [677](#)
- LPI2C\_DRV\_MasterGetBaudRate
  - LPI2C Driver, [677](#)
- LPI2C\_DRV\_MasterGetDefaultConfig
  - LPI2C Driver, [677](#)
- LPI2C\_DRV\_MasterGetTransferStatus
  - LPI2C Driver, [677](#)
- LPI2C\_DRV\_MasterIRQHandler
  - LPI2C Driver, [678](#)
- LPI2C\_DRV\_MasterInit
  - LPI2C Driver, [678](#)
- LPI2C\_DRV\_MasterReceiveData
  - LPI2C Driver, [678](#)
- LPI2C\_DRV\_MasterReceiveDataBlocking
  - LPI2C Driver, [679](#)
- LPI2C\_DRV\_MasterSendData
  - LPI2C Driver, [679](#)
- LPI2C\_DRV\_MasterSendDataBlocking
  - LPI2C Driver, [679](#)
- LPI2C\_DRV\_MasterSetBaudRate
  - LPI2C Driver, [680](#)
- LPI2C\_DRV\_MasterSetSlaveAddr
  - LPI2C Driver, [680](#)
- LPI2C\_DRV\_SetMasterBusIdleTimeout
  - LPI2C Driver, [680](#)
- LPI2C\_DRV\_SlaveAbortTransferData
  - LPI2C Driver, [681](#)
- LPI2C\_DRV\_SlaveDeinit
  - LPI2C Driver, [681](#)
- LPI2C\_DRV\_SlaveGetDefaultConfig
  - LPI2C Driver, [681](#)
- LPI2C\_DRV\_SlaveGetTransferStatus
  - LPI2C Driver, [681](#)
- LPI2C\_DRV\_SlaveIRQHandler
  - LPI2C Driver, [683](#)
- LPI2C\_DRV\_SlaveInit
  - LPI2C Driver, [681](#)
- LPI2C\_DRV\_SlaveReceiveData
  - LPI2C Driver, [683](#)
- LPI2C\_DRV\_SlaveReceiveDataBlocking
  - LPI2C Driver, [683](#)
- LPI2C\_DRV\_SlaveSendData
  - LPI2C Driver, [684](#)
- LPI2C\_DRV\_SlaveSendDataBlocking
  - LPI2C Driver, [684](#)
- LPI2C\_DRV\_SlaveSetRxBuffer
  - LPI2C Driver, [684](#)
- LPI2C\_DRV\_SlaveSetTxBuffer
  - LPI2C Driver, [685](#)
- LPI2C\_FAST\_MODE
  - LPI2C Driver, [676](#)
- LPI2C\_STANDARD\_MODE
  - LPI2C Driver, [676](#)
- LPI2C\_USING\_DMA
  - LPI2C Driver, [676](#)
- LPI2C\_USING\_INTERRUPTS
  - LPI2C Driver, [676](#)
- LPIT Driver, [686](#)
  - LPIT\_DRV\_ClearInterruptFlagTimerChannels, [692](#)
  - LPIT\_DRV\_Deinit, [693](#)
  - LPIT\_DRV\_DisableTimerChannelInterrupt, [693](#)

- LPIT\_DRV\_EnableTimerChannelInterrupt, [693](#)
- LPIT\_DRV\_GetCurrentTimerCount, [694](#)
- LPIT\_DRV\_GetCurrentTimerUs, [694](#)
- LPIT\_DRV\_GetDefaultChanConfig, [694](#)
- LPIT\_DRV\_GetDefaultConfig, [695](#)
- LPIT\_DRV\_GetInterruptFlagTimerChannels, [695](#)
- LPIT\_DRV\_GetTimerPeriodByCount, [695](#)
- LPIT\_DRV\_GetTimerPeriodByUs, [697](#)
- LPIT\_DRV\_Init, [697](#)
- LPIT\_DRV\_InitChannel, [697](#)
- LPIT\_DRV\_SetTimerPeriodByCount, [698](#)
- LPIT\_DRV\_SetTimerPeriodByUs, [698](#)
- LPIT\_DRV\_SetTimerPeriodInDual16ModeByCount, [699](#)
- LPIT\_DRV\_SetTimerPeriodInDual16ModeByUs, [699](#)
- LPIT\_DRV\_StartTimerChannels, [699](#)
- LPIT\_DRV\_StopTimerChannels, [700](#)
- LPIT\_DUAL\_PERIODIC\_COUNTER, [692](#)
- LPIT\_INPUT\_CAPTURE, [692](#)
- LPIT\_PERIOD\_UNITS\_COUNTS, [692](#)
- LPIT\_PERIOD\_UNITS\_MICROSECONDS, [692](#)
- LPIT\_PERIODIC\_COUNTER, [692](#)
- LPIT\_TRIGGER\_ACCUMULATOR, [692](#)
- LPIT\_TRIGGER\_SOURCE\_EXTERNAL, [692](#)
- LPIT\_TRIGGER\_SOURCE\_INTERNAL, [692](#)
- lpit\_period\_units\_t, [692](#)
- lpit\_timer\_modes\_t, [692](#)
- lpit\_trigger\_source\_t, [692](#)
- MAX\_PERIOD\_COUNT, [691](#)
- MAX\_PERIOD\_COUNT\_16\_BIT, [691](#)
- MAX\_PERIOD\_COUNT\_IN\_DUAL\_16BIT\_MODE, [691](#)
- LPIT\_DRV\_ClearInterruptFlagTimerChannels
  - LPIT Driver, [692](#)
- LPIT\_DRV\_Deinit
  - LPIT Driver, [693](#)
- LPIT\_DRV\_DisableTimerChannelInterrupt
  - LPIT Driver, [693](#)
- LPIT\_DRV\_EnableTimerChannelInterrupt
  - LPIT Driver, [693](#)
- LPIT\_DRV\_GetCurrentTimerCount
  - LPIT Driver, [694](#)
- LPIT\_DRV\_GetCurrentTimerUs
  - LPIT Driver, [694](#)
- LPIT\_DRV\_GetDefaultChanConfig
  - LPIT Driver, [694](#)
- LPIT\_DRV\_GetDefaultConfig
  - LPIT Driver, [695](#)
- LPIT\_DRV\_GetInterruptFlagTimerChannels
  - LPIT Driver, [695](#)
- LPIT\_DRV\_GetTimerPeriodByCount
  - LPIT Driver, [695](#)
- LPIT\_DRV\_GetTimerPeriodByUs
  - LPIT Driver, [697](#)
- LPIT\_DRV\_Init
  - LPIT Driver, [697](#)
- LPIT\_DRV\_InitChannel
  - LPIT Driver, [697](#)
- LPIT\_DRV\_SetTimerPeriodByCount
  - LPIT Driver, [698](#)
- LPIT\_DRV\_SetTimerPeriodByUs
  - LPIT Driver, [698](#)
- LPIT\_DRV\_SetTimerPeriodInDual16ModeByCount
  - LPIT Driver, [699](#)
- LPIT\_DRV\_SetTimerPeriodInDual16ModeByUs
  - LPIT Driver, [699](#)
- LPIT\_DRV\_StartTimerChannels
  - LPIT Driver, [699](#)
- LPIT\_DRV\_StopTimerChannels
  - LPIT Driver, [700](#)
- LPIT\_DUAL\_PERIODIC\_COUNTER
  - LPIT Driver, [692](#)
- LPIT\_INPUT\_CAPTURE
  - LPIT Driver, [692](#)
- LPIT\_PERIOD\_UNITS\_COUNTS
  - LPIT Driver, [692](#)
- LPIT\_PERIOD\_UNITS\_MICROSECONDS
  - LPIT Driver, [692](#)
- LPIT\_PERIODIC\_COUNTER
  - LPIT Driver, [692](#)
- LPIT\_TRIGGER\_ACCUMULATOR
  - LPIT Driver, [692](#)
- LPIT\_TRIGGER\_SOURCE\_EXTERNAL
  - LPIT Driver, [692](#)
- LPIT\_TRIGGER\_SOURCE\_INTERNAL
  - LPIT Driver, [692](#)
- LPSPi Driver, [701](#)
  - g\_lpspiBase, [718](#)
  - g\_lpspiIrqId, [718](#)
  - g\_lpspiStatePtr, [718](#)
  - LPSPi0\_IRQHandler, [710](#)
  - LPSPi1\_IRQHandler, [710](#)
  - LPSPi2\_IRQHandler, [710](#)
  - LPSPi\_ACTIVE\_HIGH, [709](#)
  - LPSPi\_ACTIVE\_LOW, [709](#)
  - LPSPi\_CLOCK\_PHASE\_1ST\_EDGE, [709](#)
  - LPSPi\_CLOCK\_PHASE\_2ND\_EDGE, [709](#)
  - LPSPi\_DRV\_DisableTEIInterrupts, [710](#)
  - LPSPi\_DRV\_FillupTxBuffer, [710](#)
  - LPSPi\_DRV\_IRQHandler, [710](#)
  - LPSPi\_DRV\_MasterAbortTransfer, [711](#)
  - LPSPi\_DRV\_MasterConfigureBus, [711](#)
  - LPSPi\_DRV\_MasterDeinit, [712](#)
  - LPSPi\_DRV\_MasterGetDefaultConfig, [712](#)
  - LPSPi\_DRV\_MasterGetTransferStatus, [712](#)
  - LPSPi\_DRV\_MasterIRQHandler, [713](#)
  - LPSPi\_DRV\_MasterInit, [712](#)
  - LPSPi\_DRV\_MasterSetDelay, [713](#)
  - LPSPi\_DRV\_MasterTransfer, [714](#)
  - LPSPi\_DRV\_MasterTransferBlocking, [714](#)
  - LPSPi\_DRV\_ReadRXBuffer, [715](#)
  - LPSPi\_DRV\_SetPcs, [715](#)
  - LPSPi\_DRV\_SlaveAbortTransfer, [715](#)
  - LPSPi\_DRV\_SlaveDeinit, [716](#)
  - LPSPi\_DRV\_SlaveGetDefaultConfig, [716](#)

- LPSPIDRV\_SlaveGetTransferStatus, 716
- LPSPIDRV\_SlaveIRQHandler, 717
- LPSPIDRV\_SlaveInit, 716
- LPSPIDRV\_SlaveTransfer, 717
- LPSPIDRV\_SlaveTransferBlocking, 717
- LPSPIDRV\_PCS0, 709
- LPSPIDRV\_PCS1, 709
- LPSPIDRV\_PCS2, 710
- LPSPIDRV\_PCS3, 710
- LPSPIDRV\_RECEIVE\_FAIL, 710
- LPSPIDRV\_SCK\_ACTIVE\_HIGH, 709
- LPSPIDRV\_SCK\_ACTIVE\_LOW, 709
- LPSPIDRV\_TRANSFER\_OK, 710
- LPSPIDRV\_TRANSMIT\_FAIL, 710
- LPSPIDRV\_USING\_DMA, 709
- LPSPIDRV\_USING\_INTERRUPTS, 709
- lpspi\_clock\_phase\_t, 709
- lpspi\_sck\_polarity\_t, 709
- lpspi\_signal\_polarity\_t, 709
- lpspi\_transfer\_type, 709
- lpspi\_which\_pcs\_t, 709
- transfer\_status\_t, 710
- LPSPIDRV\_IRQHandler
  - LPSPIDRV Driver, 710
- LPSPIDRV\_IRQHandler
  - LPSPIDRV Driver, 710
- LPSPIDRV\_IRQHandler
  - LPSPIDRV Driver, 710
- LPSPIDRV\_ACTIVE\_HIGH
  - LPSPIDRV Driver, 709
- LPSPIDRV\_ACTIVE\_LOW
  - LPSPIDRV Driver, 709
- LPSPIDRV\_CLOCK\_PHASE\_1ST\_EDGE
  - LPSPIDRV Driver, 709
- LPSPIDRV\_CLOCK\_PHASE\_2ND\_EDGE
  - LPSPIDRV Driver, 709
- LPSPIDRV\_DisableTEIEInterrupts
  - LPSPIDRV Driver, 710
- LPSPIDRV\_FillupTxBuffer
  - LPSPIDRV Driver, 710
- LPSPIDRV\_IRQHandler
  - LPSPIDRV Driver, 710
- LPSPIDRV\_MasterAbortTransfer
  - LPSPIDRV Driver, 711
- LPSPIDRV\_MasterConfigureBus
  - LPSPIDRV Driver, 711
- LPSPIDRV\_MasterDeinit
  - LPSPIDRV Driver, 712
- LPSPIDRV\_MasterGetDefaultConfig
  - LPSPIDRV Driver, 712
- LPSPIDRV\_MasterGetTransferStatus
  - LPSPIDRV Driver, 712
- LPSPIDRV\_MasterIRQHandler
  - LPSPIDRV Driver, 713
- LPSPIDRV\_MasterInit
  - LPSPIDRV Driver, 712
- LPSPIDRV\_MasterSetDelay
  - LPSPIDRV Driver, 713
- LPSPIDRV\_MasterTransfer
  - LPSPIDRV Driver, 714
- LPSPIDRV\_MasterTransferBlocking
  - LPSPIDRV Driver, 714
- LPSPIDRV\_ReadRXBuffer
  - LPSPIDRV Driver, 715
- LPSPIDRV\_SetPcs
  - LPSPIDRV Driver, 715
- LPSPIDRV\_SlaveAbortTransfer
  - LPSPIDRV Driver, 715
- LPSPIDRV\_SlaveDeinit
  - LPSPIDRV Driver, 716
- LPSPIDRV\_SlaveGetDefaultConfig
  - LPSPIDRV Driver, 716
- LPSPIDRV\_SlaveGetTransferStatus
  - LPSPIDRV Driver, 716
- LPSPIDRV\_SlaveIRQHandler
  - LPSPIDRV Driver, 717
- LPSPIDRV\_SlaveInit
  - LPSPIDRV Driver, 716
- LPSPIDRV\_SlaveTransfer
  - LPSPIDRV Driver, 717
- LPSPIDRV\_SlaveTransferBlocking
  - LPSPIDRV Driver, 717
- LPSPIDRV\_PCS0
  - LPSPIDRV Driver, 709
- LPSPIDRV\_PCS1
  - LPSPIDRV Driver, 709
- LPSPIDRV\_PCS2
  - LPSPIDRV Driver, 710
- LPSPIDRV\_PCS3
  - LPSPIDRV Driver, 710
- LPSPIDRV\_RECEIVE\_FAIL
  - LPSPIDRV Driver, 710
- LPSPIDRV\_SCK\_ACTIVE\_HIGH
  - LPSPIDRV Driver, 709
- LPSPIDRV\_SCK\_ACTIVE\_LOW
  - LPSPIDRV Driver, 709
- LPSPIDRV\_TRANSFER\_OK
  - LPSPIDRV Driver, 710
- LPSPIDRV\_TRANSMIT\_FAIL
  - LPSPIDRV Driver, 710
- LPSPIDRV\_USING\_DMA
  - LPSPIDRV Driver, 709
- LPSPIDRV\_USING\_INTERRUPTS
  - LPSPIDRV Driver, 709
- LPTMR Driver, 719
  - LPTMR\_CLOCKSOURCE\_1KHZ\_LPO, 723
  - LPTMR\_CLOCKSOURCE\_PCC, 723
  - LPTMR\_CLOCKSOURCE\_RTC, 723
  - LPTMR\_CLOCKSOURCE\_SIRCDIV2, 723
  - LPTMR\_COUNTER\_UNITS\_MICROSECONDS, 723
  - LPTMR\_COUNTER\_UNITS\_TICKS, 723
  - LPTMR\_DRV\_ClearCompareFlag, 725
  - LPTMR\_DRV\_Deinit, 725
  - LPTMR\_DRV\_GetCompareFlag, 725
  - LPTMR\_DRV\_GetCompareValueByCount, 725

- LPTMR\_DRV\_GetCompareValueByUs, [725](#)
- LPTMR\_DRV\_GetConfig, [726](#)
- LPTMR\_DRV\_GetCounterValueByCount, [726](#)
- LPTMR\_DRV\_Init, [726](#)
- LPTMR\_DRV\_InitConfigStruct, [726](#)
- LPTMR\_DRV\_IsRunning, [726](#)
- LPTMR\_DRV\_SetCompareValueByCount, [727](#)
- LPTMR\_DRV\_SetCompareValueByUs, [727](#)
- LPTMR\_DRV\_SetConfig, [727](#)
- LPTMR\_DRV\_SetInterrupt, [728](#)
- LPTMR\_DRV\_SetPinConfiguration, [728](#)
- LPTMR\_DRV\_StartCounter, [728](#)
- LPTMR\_DRV\_StopCounter, [728](#)
- LPTMR\_PINPOLARITY\_FALLING, [724](#)
- LPTMR\_PINPOLARITY\_RISING, [724](#)
- LPTMR\_PINSELECT\_ALT2, [724](#)
- LPTMR\_PINSELECT\_ALT3, [724](#)
- LPTMR\_PINSELECT\_TRGMUX, [724](#)
- LPTMR\_PRESCALE\_1024\_GLITCHFILTER\_512, [724](#)
- LPTMR\_PRESCALE\_128\_GLITCHFILTER\_64, [724](#)
- LPTMR\_PRESCALE\_16384\_GLITCHFILTER\_↔ 8192, [724](#)
- LPTMR\_PRESCALE\_16\_GLITCHFILTER\_8, [724](#)
- LPTMR\_PRESCALE\_2, [724](#)
- LPTMR\_PRESCALE\_2048\_GLITCHFILTER\_↔ 1024, [724](#)
- LPTMR\_PRESCALE\_256\_GLITCHFILTER\_128, [724](#)
- LPTMR\_PRESCALE\_32768\_GLITCHFILTER\_↔ 16384, [724](#)
- LPTMR\_PRESCALE\_32\_GLITCHFILTER\_16, [724](#)
- LPTMR\_PRESCALE\_4096\_GLITCHFILTER\_↔ 2048, [724](#)
- LPTMR\_PRESCALE\_4\_GLITCHFILTER\_2, [724](#)
- LPTMR\_PRESCALE\_512\_GLITCHFILTER\_256, [724](#)
- LPTMR\_PRESCALE\_64\_GLITCHFILTER\_32, [724](#)
- LPTMR\_PRESCALE\_65536\_GLITCHFILTER\_↔ 32768, [724](#)
- LPTMR\_PRESCALE\_8192\_GLITCHFILTER\_↔ 4096, [724](#)
- LPTMR\_PRESCALE\_8\_GLITCHFILTER\_4, [724](#)
- LPTMR\_WORKMODE\_PULSECOUNTER, [725](#)
- LPTMR\_WORKMODE\_TIMER, [725](#)
- lptmr\_clocksource\_t, [723](#)
- lptmr\_counter\_units\_t, [723](#)
- lptmr\_pinpolarity\_t, [723](#)
- lptmr\_pinselect\_t, [724](#)
- lptmr\_prescaler\_t, [724](#)
- lptmr\_workmode\_t, [724](#)
- LPTMR\_CLOCKSOURCE\_1KHZ\_LPO
  - LPTMR Driver, [723](#)
- LPTMR\_CLOCKSOURCE\_PCC
  - LPTMR Driver, [723](#)
- LPTMR\_CLOCKSOURCE\_RTC
  - LPTMR Driver, [723](#)
- LPTMR\_CLOCKSOURCE\_SIRCDIV2
  - LPTMR Driver, [723](#)
- LPTMR\_COUNTER\_UNITS\_MICROSECONDS
  - LPTMR Driver, [723](#)
- LPTMR\_COUNTER\_UNITS\_TICKS
  - LPTMR Driver, [723](#)
- LPTMR\_DRV\_ClearCompareFlag
  - LPTMR Driver, [725](#)
- LPTMR\_DRV\_Deinit
  - LPTMR Driver, [725](#)
- LPTMR\_DRV\_GetCompareFlag
  - LPTMR Driver, [725](#)
- LPTMR\_DRV\_GetCompareValueByCount
  - LPTMR Driver, [725](#)
- LPTMR\_DRV\_GetCompareValueByUs
  - LPTMR Driver, [725](#)
- LPTMR\_DRV\_GetConfig
  - LPTMR Driver, [726](#)
- LPTMR\_DRV\_GetCounterValueByCount
  - LPTMR Driver, [726](#)
- LPTMR\_DRV\_Init
  - LPTMR Driver, [726](#)
- LPTMR\_DRV\_InitConfigStruct
  - LPTMR Driver, [726](#)
- LPTMR\_DRV\_IsRunning
  - LPTMR Driver, [726](#)
- LPTMR\_DRV\_SetCompareValueByCount
  - LPTMR Driver, [727](#)
- LPTMR\_DRV\_SetCompareValueByUs
  - LPTMR Driver, [727](#)
- LPTMR\_DRV\_SetConfig
  - LPTMR Driver, [727](#)
- LPTMR\_DRV\_SetInterrupt
  - LPTMR Driver, [728](#)
- LPTMR\_DRV\_SetPinConfiguration
  - LPTMR Driver, [728](#)
- LPTMR\_DRV\_StartCounter
  - LPTMR Driver, [728](#)
- LPTMR\_DRV\_StopCounter
  - LPTMR Driver, [728](#)
- LPTMR\_PINPOLARITY\_FALLING
  - LPTMR Driver, [724](#)
- LPTMR\_PINPOLARITY\_RISING
  - LPTMR Driver, [724](#)
- LPTMR\_PINSELECT\_ALT2
  - LPTMR Driver, [724](#)
- LPTMR\_PINSELECT\_ALT3
  - LPTMR Driver, [724](#)
- LPTMR\_PINSELECT\_TRGMUX
  - LPTMR Driver, [724](#)
- LPTMR\_PRESCALE\_1024\_GLITCHFILTER\_512
  - LPTMR Driver, [724](#)
- LPTMR\_PRESCALE\_128\_GLITCHFILTER\_64
  - LPTMR Driver, [724](#)
- LPTMR\_PRESCALE\_16384\_GLITCHFILTER\_8192
  - LPTMR Driver, [724](#)
- LPTMR\_PRESCALE\_16\_GLITCHFILTER\_8
  - LPTMR Driver, [724](#)



LPTMR\_PRESCALE\_2  
LPTMR Driver, [724](#)

LPTMR\_PRESCALE\_2048\_GLITCHFILTER\_1024  
LPTMR Driver, [724](#)

LPTMR\_PRESCALE\_256\_GLITCHFILTER\_128  
LPTMR Driver, [724](#)

LPTMR\_PRESCALE\_32768\_GLITCHFILTER\_16384  
LPTMR Driver, [724](#)

LPTMR\_PRESCALE\_32\_GLITCHFILTER\_16  
LPTMR Driver, [724](#)

LPTMR\_PRESCALE\_4096\_GLITCHFILTER\_2048  
LPTMR Driver, [724](#)

LPTMR\_PRESCALE\_4\_GLITCHFILTER\_2  
LPTMR Driver, [724](#)

LPTMR\_PRESCALE\_512\_GLITCHFILTER\_256  
LPTMR Driver, [724](#)

LPTMR\_PRESCALE\_64\_GLITCHFILTER\_32  
LPTMR Driver, [724](#)

LPTMR\_PRESCALE\_65536\_GLITCHFILTER\_32768  
LPTMR Driver, [724](#)

LPTMR\_PRESCALE\_8192\_GLITCHFILTER\_4096  
LPTMR Driver, [724](#)

LPTMR\_PRESCALE\_8\_GLITCHFILTER\_4  
LPTMR Driver, [724](#)

LPTMR\_WORKMODE\_PULSECOUNTER  
LPTMR Driver, [725](#)

LPTMR\_WORKMODE\_TIMER  
LPTMR Driver, [725](#)

LPUART Driver, [729](#)  
LPUART\_10\_BITS\_PER\_CHAR, [736](#)  
LPUART\_8\_BITS\_PER\_CHAR, [736](#)  
LPUART\_9\_BITS\_PER\_CHAR, [736](#)  
LPUART\_DRV\_AbortReceivingData, [736](#)  
LPUART\_DRV\_AbortSendingData, [737](#)  
LPUART\_DRV\_Deinit, [737](#)  
LPUART\_DRV\_GetBaudRate, [737](#)  
LPUART\_DRV\_GetDefaultConfig, [737](#)  
LPUART\_DRV\_GetReceiveStatus, [738](#)  
LPUART\_DRV\_GetTransmitStatus, [738](#)  
LPUART\_DRV\_Init, [739](#)  
LPUART\_DRV\_InstallRxCallback, [739](#)  
LPUART\_DRV\_InstallTxCallback, [739](#)  
LPUART\_DRV\_ReceiveData, [740](#)  
LPUART\_DRV\_ReceiveDataBlocking, [740](#)  
LPUART\_DRV\_ReceiveDataPolling, [740](#)  
LPUART\_DRV\_SendData, [741](#)  
LPUART\_DRV\_SendDataBlocking, [741](#)  
LPUART\_DRV\_SendDataPolling, [741](#)  
LPUART\_DRV\_SetBaudRate, [742](#)  
LPUART\_DRV\_SetRxBuffer, [742](#)  
LPUART\_DRV\_SetTxBuffer, [742](#)  
LPUART\_ONE\_STOP\_BIT, [736](#)  
LPUART\_PARITY\_DISABLED, [736](#)  
LPUART\_PARITY\_EVEN, [736](#)  
LPUART\_PARITY\_ODD, [736](#)  
LPUART\_TWO\_STOP\_BIT, [736](#)  
LPUART\_USING\_DMA, [736](#)  
LPUART\_USING\_INTERRUPTS, [736](#)  
lpuart\_bit\_count\_per\_char\_t, [736](#)  
lpuart\_parity\_mode\_t, [736](#)  
lpuart\_stop\_bit\_count\_t, [736](#)  
lpuart\_transfer\_type\_t, [736](#)

LPUART\_10\_BITS\_PER\_CHAR  
LPUART Driver, [736](#)

LPUART\_8\_BITS\_PER\_CHAR  
LPUART Driver, [736](#)

LPUART\_9\_BITS\_PER\_CHAR  
LPUART Driver, [736](#)

LPUART\_DRV\_AbortReceivingData  
LPUART Driver, [736](#)

LPUART\_DRV\_AbortSendingData  
LPUART Driver, [737](#)

LPUART\_DRV\_Deinit  
LPUART Driver, [737](#)

LPUART\_DRV\_GetBaudRate  
LPUART Driver, [737](#)

LPUART\_DRV\_GetDefaultConfig  
LPUART Driver, [737](#)

LPUART\_DRV\_GetReceiveStatus  
LPUART Driver, [738](#)

LPUART\_DRV\_GetTransmitStatus  
LPUART Driver, [738](#)

LPUART\_DRV\_Init  
LPUART Driver, [739](#)

LPUART\_DRV\_InstallRxCallback  
LPUART Driver, [739](#)

LPUART\_DRV\_InstallTxCallback  
LPUART Driver, [739](#)

LPUART\_DRV\_ReceiveData  
LPUART Driver, [740](#)

LPUART\_DRV\_ReceiveDataBlocking  
LPUART Driver, [740](#)

LPUART\_DRV\_ReceiveDataPolling  
LPUART Driver, [740](#)

LPUART\_DRV\_SendData  
LPUART Driver, [741](#)

LPUART\_DRV\_SendDataBlocking  
LPUART Driver, [741](#)

LPUART\_DRV\_SendDataPolling  
LPUART Driver, [741](#)

LPUART\_DRV\_SetBaudRate  
LPUART Driver, [742](#)

LPUART\_DRV\_SetRxBuffer  
LPUART Driver, [742](#)

LPUART\_DRV\_SetTxBuffer  
LPUART Driver, [742](#)

LPUART\_ONE\_STOP\_BIT  
LPUART Driver, [736](#)

LPUART\_PARITY\_DISABLED  
LPUART Driver, [736](#)

LPUART\_PARITY\_EVEN  
LPUART Driver, [736](#)

LPUART\_PARITY\_ODD  
LPUART Driver, [736](#)

LPUART\_TWO\_STOP\_BIT  
LPUART Driver, [736](#)

- LPUART\_USING\_DMA
  - LPUART Driver, [736](#)
- LPUART\_USING\_INTERRUPTS
  - LPUART Driver, [736](#)
- LSC1\_SLOPE\_ACTIVE
  - 02. LIN configuration, [171](#)
- LSC1\_SLOPE\_ACTIVE2
  - 02. LIN configuration, [171](#)
- LSC1\_SLOPE\_ACTIVE\_TXDL\_TIMEOUT
  - 02. LIN configuration, [171](#)
- LSC2\_SLOPE\_ACTIVE
  - 02. LIN configuration, [171](#)
- LSC2\_SLOPE\_ACTIVE2
  - 02. LIN configuration, [171](#)
- LSC2\_SLOPE\_ACTIVE\_TXDL\_TIMEOUT
  - 02. LIN configuration, [171](#)
- language\_version
  - lin\_protocol\_user\_config\_t, [766](#)
- last\_RSID
  - lin\_tl\_descriptor\_t, [764](#)
- last\_cfg\_result
  - lin\_tl\_descriptor\_t, [764](#)
- last\_pid
  - lin\_protocol\_state\_t, [770](#)
  - lin\_word\_status\_str\_t, [756](#)
- lckcLockRegCfg
  - uja113x\_init\_cfg\_t, [145](#)
- ld\_assign\_NAD
  - Node configuration, [818](#)
- ld\_assign\_NAD\_j2602
  - Node configuration, [816](#)
- ld\_assign\_frame\_id\_range
  - Node configuration, [818](#)
- ld\_check\_response
  - Node configuration, [820](#)
- ld\_check\_response\_j2602
  - Node configuration, [816](#)
- ld\_conditional\_change\_NAD
  - Node configuration, [820](#)
- ld\_error\_code
  - lin\_tl\_descriptor\_t, [764](#)
- ld\_get\_raw
  - Raw API, [908](#)
- ld\_init
  - Initialization, [610](#)
- ld\_is\_ready
  - Node configuration, [820](#)
- ld\_is\_ready\_j2602
  - Node configuration, [816](#)
- ld\_put\_raw
  - Raw API, [908](#)
- ld\_queue\_status\_t
  - Low level API, [774](#)
- ld\_raw\_rx\_status
  - Raw API, [908](#)
- ld\_raw\_tx\_status
  - Raw API, [909](#)
- ld\_read\_by\_id
  - Node identification, [823](#)
- ld\_read\_by\_id\_callout
  - Low level API, [778](#)
- ld\_read\_configuration
  - Node configuration, [821](#)
- ld\_receive\_message
  - Cooked API, [361](#)
- ld\_reconfig\_msg\_ID
  - Node configuration, [817](#)
- ld\_return\_data
  - lin\_tl\_descriptor\_t, [764](#)
- ld\_rx\_status
  - Cooked API, [361](#)
- ld\_save\_configuration
  - Node configuration, [821](#)
- ld\_send\_message
  - Cooked API, [362](#)
- ld\_set\_configuration
  - Node configuration, [821](#)
- ld\_tx\_status
  - Cooked API, [362](#)
- length
  - can\_message\_t, [349](#)
  - edma\_scatter\_gather\_list\_t, [378](#)
- lhc
  - sbc\_int\_config\_t, [1020](#)
- limpHomeCtrl
  - uja113x\_fsc\_cfg\_t, [179](#)
- lin1ModeCtrl
  - uja113x\_linc\_cfg\_t, [170](#)
- lin1SlopeCtrl
  - uja113x\_linc\_cfg\_t, [170](#)
- lin2ModeCtrl
  - uja113x\_linc\_cfg\_t, [170](#)
- lin2SlopeCtrl
  - uja113x\_linc\_cfg\_t, [170](#)
- lin\_associate\_frame\_t, [759](#)
  - associated\_uncond\_frame\_ptr, [759](#)
  - coll\_resolv\_schd, [759](#)
  - num\_of\_associated\_uncond\_frames, [760](#)
- lin\_calc\_max\_header\_timeout\_cnt
  - Low level API, [778](#)
- lin\_calc\_max\_res\_timeout\_cnt
  - Low level API, [778](#)
- lin\_callback\_t
  - LIN Driver, [657](#)
- lin\_collision\_resolve
  - LIN 2.1 Specific API, [645](#)
- lin\_diag\_service\_callback
  - Common Transport Layer API, [321](#)
- lin\_diagnostic\_class\_t
  - Low level API, [774](#)
- lin\_diagnostic\_state\_t
  - Low level API, [774](#)
- lin\_event\_id\_t
  - LIN Driver, [657](#)
- lin\_frame\_response\_t
  - Low level API, [775](#)



- lin\_frame\_t, 760
  - flag\_offset, 760
  - flag\_size, 760
  - frame\_data\_ptr, 760
  - frm\_len, 760
  - frm\_offset, 760
  - frm\_response, 760
  - frm\_type, 760
- lin\_frame\_type\_t
  - Low level API, 775
- lin\_last\_cfg\_result\_t
  - Low level API, 775
- lin\_ild\_deinit
  - Low level API, 778
- lin\_ild\_event\_id\_t
  - Low level API, 775
- lin\_ild\_get\_state
  - Low level API, 778
- lin\_ild\_ignore\_response
  - Low level API, 779
- lin\_ild\_init
  - Low level API, 779
- lin\_ild\_int\_disable
  - Low level API, 779
- lin\_ild\_int\_enable
  - Low level API, 779
- lin\_ild\_rx\_response
  - Low level API, 780
- lin\_ild\_set\_low\_power\_mode
  - Low level API, 780
- lin\_ild\_set\_response
  - Low level API, 780
- lin\_ild\_timeout\_service
  - Low level API, 780
- lin\_ild\_tx\_header
  - Low level API, 781
- lin\_ild\_tx\_wake\_up
  - Low level API, 781
- lin\_make\_res\_evnt\_frame
  - LIN 2.1 Specific API, 645
- lin\_master\_data\_t, 767
  - active\_schedule\_id, 768
  - event\_trigger\_collision\_flg, 768
  - flag\_offset, 768
  - flag\_size, 768
  - frm\_offset, 768
  - frm\_size, 768
  - master\_data\_buffer, 768
  - previous\_schedule\_id, 768
  - schedule\_start\_entry\_ptr, 768
  - send\_functional\_request\_flg, 769
  - send\_slave\_res\_flg, 769
- lin\_message\_status\_t
  - Low level API, 776
- lin\_message\_timeout\_type\_t
  - Low level API, 776
- lin\_node\_attribute\_t, 757
  - configured\_NAD\_ptr, 758
  - fault\_state\_signal\_ptr, 758
  - initial\_NAD, 758
  - N\_As\_timeout, 758
  - N\_Cr\_timeout, 758
  - num\_frame\_have\_esignal, 758
  - num\_of\_fault\_state\_signal, 758
  - number\_support\_sid, 758
  - P2\_min, 758
  - product\_id, 758
  - resp\_err\_frm\_id\_ptr, 758
  - response\_error, 759
  - response\_error\_bit\_offset\_ptr, 759
  - response\_error\_byte\_offset\_ptr, 759
  - ST\_min, 759
  - serial\_number, 759
  - service\_flags\_ptr, 759
  - service\_supported\_ptr, 759
- lin\_node\_state\_t
  - LIN Driver, 658
- lin\_pid\_resp\_callback\_handler
  - Low level API, 781
- lin\_process\_parity
  - Low level API, 782
- lin\_product\_id\_t, 1085
  - function\_id, 1085
  - supplier\_id, 1085
  - variant, 1085
- lin\_protocol\_handle\_t
  - Low level API, 776
- lin\_protocol\_state\_t, 769
  - baud\_rate, 769
  - current\_id, 769
  - diagnostic\_mode, 769
  - error\_in\_response, 769
  - frame\_timeout\_cnt, 770
  - go\_to\_sleep\_flg, 770
  - idle\_timeout\_cnt, 770
  - last\_pid, 770
  - next\_transmit\_tick, 770
  - num\_of\_processed\_frame, 770
  - overrun\_flg, 770
  - response\_buffer\_ptr, 770
  - response\_length, 770
  - save\_config\_flg, 770
  - successful\_transfer, 770
  - transmit\_error\_resp\_sig\_flg, 771
  - word\_status, 771
- lin\_protocol\_user\_config\_t, 765
  - diagnostic\_class, 766
  - frame\_start, 766
  - frame\_tbl\_ptr, 766
  - function, 766
  - language\_version, 766
  - lin\_user\_config\_ptr, 766
  - list\_identifiers\_RAM\_ptr, 766
  - list\_identifiers\_ROM\_ptr, 766
  - master\_ifc\_handle, 766
  - max\_idle\_timeout\_cnt, 767

- max\_message\_length, [767](#)
- num\_of\_schedules, [767](#)
- number\_of\_configurable\_frames, [767](#)
- protocol\_version, [767](#)
- schedule\_start, [767](#)
- schedule\_tbl, [767](#)
- slave\_ifc\_handle, [767](#)
- tl\_rx\_queue\_data\_ptr, [767](#)
- tl\_tx\_queue\_data\_ptr, [767](#)
- lin\_sch\_tbl\_type\_t
  - Low level API, [776](#)
- lin\_schedule\_data\_t, [761](#)
  - delay\_integer, [761](#)
  - frm\_id, [761](#)
  - tl\_queue\_data, [761](#)
- lin\_schedule\_t, [761](#)
  - num\_slots, [761](#)
  - ptr\_sch\_data\_ptr, [761](#)
  - sch\_tbl\_type, [761](#)
- lin\_serial\_number\_t, [757](#)
  - serial\_0, [757](#)
  - serial\_1, [757](#)
  - serial\_2, [757](#)
  - serial\_3, [757](#)
- lin\_service\_status\_t
  - Low level API, [777](#)
- lin\_state\_t, [654](#)
  - baudrateEvalEnable, [655](#)
  - Callback, [655](#)
  - checksum, [655](#)
  - cntByte, [655](#)
  - currentEventId, [655](#)
  - currentId, [655](#)
  - currentNodeState, [655](#)
  - currentPid, [655](#)
  - fallingEdgeInterruptCount, [655](#)
  - isBusBusy, [655](#)
  - isRxBlocking, [655](#)
  - isRxBusy, [655](#)
  - isTxBlocking, [656](#)
  - isTxBusy, [656](#)
  - linSourceClockFreq, [656](#)
  - rxBuff, [656](#)
  - rxCompleted, [656](#)
  - rxSize, [656](#)
  - timeoutCounter, [656](#)
  - timeoutCounterFlag, [656](#)
  - txBuff, [656](#)
  - txCompleted, [656](#)
  - txSize, [656](#)
- lin\_timer\_get\_time\_interval\_t
  - LIN Driver, [657](#)
- lin\_tl\_callback\_handler
  - Low level API, [782](#)
- lin\_tl\_callback\_return\_t
  - Low level API, [777](#)
- lin\_tl\_descriptor\_t, [762](#)
  - check\_timeout, [763](#)
  - check\_timeout\_type, [763](#)
  - diag\_interleave\_state, [763](#)
  - diag\_state, [763](#)
  - FF\_pdu\_received, [763](#)
  - frame\_counter, [763](#)
  - interleave\_timeout\_counter, [764](#)
  - last\_RSID, [764](#)
  - last\_cfg\_result, [764](#)
  - ld\_error\_code, [764](#)
  - ld\_return\_data, [764](#)
  - num\_of\_pdu, [764](#)
  - product\_id\_ptr, [764](#)
  - receive\_NAD\_ptr, [764](#)
  - receive\_message\_length\_ptr, [764](#)
  - receive\_message\_ptr, [764](#)
  - rx\_msg\_size, [764](#)
  - rx\_msg\_status, [765](#)
  - service\_status, [765](#)
  - slave\_resp\_cnt, [765](#)
  - tl\_rx\_queue, [765](#)
  - tl\_tx\_queue, [765](#)
  - tx\_msg\_size, [765](#)
  - tx\_msg\_status, [765](#)
- lin\_tl\_event\_id\_t
  - Low level API, [777](#)
- lin\_tl\_pdu\_data\_t
  - Low level API, [773](#)
- lin\_tl\_queue\_t
  - Low level API, [773](#)
- lin\_transport\_layer\_queue\_t, [762](#)
  - queue\_current\_size, [762](#)
  - queue\_header, [762](#)
  - queue\_max\_size, [762](#)
  - queue\_status, [762](#)
  - queue\_tail, [762](#)
  - tl\_pdu\_ptr, [762](#)
- lin\_update\_err\_signal
  - LIN 2.1 Specific API, [646](#)
- lin\_update\_rx\_evnt\_frame
  - LIN 2.1 Specific API, [646](#)
- lin\_update\_word\_status\_lin21
  - LIN 2.1 Specific API, [646](#)
- lin\_user\_config\_ptr
  - lin\_protocol\_user\_config\_t, [766](#)
- lin\_user\_config\_t, [653](#)
  - autobaudEnable, [653](#)
  - baudRate, [653](#)
  - classicPID, [653](#)
  - nodeFunction, [654](#)
  - numOfClassicPID, [654](#)
  - timerGetTimeIntervalCallback, [654](#)
- lin\_word\_status\_str\_t, [755](#)
  - bus\_activity, [756](#)
  - error\_in\_res, [756](#)
  - event\_trigger\_collision\_flg, [756](#)
  - go\_to\_sleep\_flg, [756](#)
  - last\_pid, [756](#)
  - overrun, [756](#)

- reserved, 756
- save\_config\_flg, 756
- successful\_transfer, 756
- linSourceClockFreq
  - lin\_state\_t, 656
- linxLinCtrl
  - uja113x\_init\_cfg\_t, 146
- list\_identifiers\_RAM\_ptr
  - lin\_protocol\_user\_config\_t, 766
- list\_identifiers\_ROM\_ptr
  - lin\_protocol\_user\_config\_t, 766
- loadValueMode
  - pdb\_timer\_config\_t, 849
- Local Interconnect Network (LIN), 744
- lock0
  - uja113x\_lckc\_cfg\_t, 194
- lock1
  - uja113x\_lckc\_cfg\_t, 194
- lock2
  - uja113x\_lckc\_cfg\_t, 194
- lock3
  - uja113x\_lckc\_cfg\_t, 194
- lock4
  - uja113x\_lckc\_cfg\_t, 194
- lock5
  - uja113x\_lckc\_cfg\_t, 194
- lock6
  - uja113x\_lckc\_cfg\_t, 194
- lockMask
  - sbc\_int\_config\_t, 1020
- lockRegisterLock
  - rtc\_register\_lock\_config\_t, 899
- lockTargetModuleReg
  - trgmux\_inout\_mapping\_config\_t, 986
- locked
  - scg\_firc\_config\_t, 292
  - scg\_sirc\_config\_t, 291
  - scg\_sosc\_config\_t, 290
  - scg\_spill\_config\_t, 293
- loopTransferConfig
  - edma\_transfer\_config\_t, 381
- Low level API, 752
  - CALLBACK\_HANDLER, 771
  - DIAG\_INTERLEAVE\_MODE, 774
  - DIAG\_NO\_RESPONSE, 774
  - DIAG\_NONE, 774
  - DIAG\_NOT\_START, 774
  - DIAG\_ONLY\_MODE, 774
  - DIAG\_RESPONSE, 774
  - diag\_interleaved\_state\_t, 773
  - g\_buffer\_backup\_data, 782
  - g\_lin\_flag\_handle\_tbl, 782
  - g\_lin\_frame\_data\_buffer, 782
  - g\_lin\_frame\_flag\_handle\_tbl, 782
  - g\_lin\_frame\_updating\_flag\_tbl, 782
  - g\_lin\_hardware\_ifc, 782
  - g\_lin\_master\_data\_array, 782
  - g\_lin\_node\_attribute\_array, 782
  - g\_lin\_protocol\_state\_array, 782
  - g\_lin\_protocol\_user\_cfg\_array, 782
  - g\_lin\_tl\_descriptor\_array, 782
  - g\_lin\_virtual\_ifc, 783
  - l\_diagnostic\_mode\_t, 774
  - LD\_CHECK\_N\_AS\_TIMEOUT, 776
  - LD\_CHECK\_N\_CR\_TIMEOUT, 776
  - LD\_COMPLETED, 776
  - LD\_DATA\_AVAILABLE, 774
  - LD\_DIAG\_IDLE, 775
  - LD\_DIAG\_RX\_FUNCTIONAL, 775
  - LD\_DIAG\_RX\_INTERLEAVED, 775
  - LD\_DIAG\_RX\_PHY, 775
  - LD\_DIAG\_TX\_FUNCTIONAL, 775
  - LD\_DIAG\_TX\_INTERLEAVED, 775
  - LD\_DIAG\_TX\_PHY, 775
  - LD\_FAILED, 776
  - LD\_IN\_PROGRESS, 776
  - LD\_N\_AS\_TIMEOUT, 776
  - LD\_N\_CR\_TIMEOUT, 776
  - LD\_NEGATIVE, 775
  - LD\_NEGATIVE\_RESPONSE, 771
  - LD\_NO\_CHECK\_TIMEOUT, 776
  - LD\_NO\_DATA, 774
  - LD\_NO\_MSG, 776
  - LD\_NO\_RESPONSE, 775
  - LD\_OVERWRITTEN, 775
  - LD\_POSITIVE\_RESPONSE, 771
  - LD\_QUEUE\_AVAILABLE, 774
  - LD\_QUEUE\_EMPTY, 774
  - LD\_QUEUE\_FULL, 774
  - LD\_RECEIVE\_ERROR, 774
  - LD\_REQUEST\_FINISHED, 777
  - LD\_SERVICE\_BUSY, 777
  - LD\_SERVICE\_ERROR, 777
  - LD\_SERVICE\_IDLE, 777
  - LD\_SUCCESS, 775
  - LD\_TRANSFER\_ERROR, 774
  - LD\_TRANSMIT\_ERROR, 774
  - LD\_WRONG\_SN, 776
  - LIN\_DIAGNOSTIC\_CLASS\_I, 774
  - LIN\_DIAGNOSTIC\_CLASS\_II, 774
  - LIN\_DIAGNOSTIC\_CLASS\_III, 774
  - LIN\_FRM\_DIAG, 775
  - LIN\_FRM\_EVNT, 775
  - LIN\_FRM\_SPRDC, 775
  - LIN\_FRM\_UNCD, 775
  - LIN\_LLD\_BUS\_ACTIVITY\_TIMEOUT, 776
  - LIN\_LLD\_CHECKSUM\_ERR, 776
  - LIN\_LLD\_ERROR, 771
  - LIN\_LLD\_FRAME\_ERR, 776
  - LIN\_LLD\_NODATA\_TIMEOUT, 776
  - LIN\_LLD\_OK, 771
  - LIN\_LLD\_PID\_ERR, 776
  - LIN\_LLD\_PID\_OK, 776
  - LIN\_LLD\_READBACK\_ERR, 776
  - LIN\_LLD\_RX\_COMPLETED, 776
  - LIN\_LLD\_TX\_COMPLETED, 776

- LIN\_MASTER, 771
- LIN\_PROTOCOL\_13, 776
- LIN\_PROTOCOL\_20, 776
- LIN\_PROTOCOL\_21, 776
- LIN\_PROTOCOL\_J2602, 776
- LIN\_READ\_USR\_DEF\_MAX, 771
- LIN\_READ\_USR\_DEF\_MIN, 771
- LIN\_RES\_PUB, 775
- LIN\_RES\_SUB, 775
- LIN\_SCH\_TBL\_COLL\_RESOLV, 777
- LIN\_SCH\_TBL\_DIAG, 777
- LIN\_SCH\_TBL\_GO\_TO\_SLEEP, 777
- LIN\_SCH\_TBL\_NORM, 777
- LIN\_SCH\_TBL\_NULL, 777
- LIN\_SLAVE, 772
- LIN\_TL\_CALLBACK\_HANDLER, 772
- ld\_queue\_status\_t, 774
- ld\_read\_by\_id\_callout, 778
- lin\_calc\_max\_header\_timeout\_cnt, 778
- lin\_calc\_max\_res\_timeout\_cnt, 778
- lin\_diagnostic\_class\_t, 774
- lin\_diagnostic\_state\_t, 774
- lin\_frame\_response\_t, 775
- lin\_frame\_type\_t, 775
- lin\_last\_cfg\_result\_t, 775
- lin\_llc\_deinit, 778
- lin\_llc\_event\_id\_t, 775
- lin\_llc\_get\_state, 778
- lin\_llc\_ignore\_response, 779
- lin\_llc\_init, 779
- lin\_llc\_int\_disable, 779
- lin\_llc\_int\_enable, 779
- lin\_llc\_rx\_response, 780
- lin\_llc\_set\_low\_power\_mode, 780
- lin\_llc\_set\_response, 780
- lin\_llc\_timeout\_service, 780
- lin\_llc\_tx\_header, 781
- lin\_llc\_tx\_wake\_up, 781
- lin\_message\_status\_t, 776
- lin\_message\_timeout\_type\_t, 776
- lin\_pid\_resp\_callback\_handler, 781
- lin\_process\_parity, 782
- lin\_protocol\_handle\_t, 776
- lin\_sch\_tbl\_type\_t, 776
- lin\_service\_status\_t, 777
- lin\_tl\_callback\_handler, 782
- lin\_tl\_callback\_return\_t, 777
- lin\_tl\_event\_id\_t, 777
- lin\_tl\_pdu\_data\_t, 773
- lin\_tl\_queue\_t, 773
- PCI\_RES\_ASSIGN\_FRAME\_ID\_RANGE, 772
- PCI\_RES\_READ\_BY\_IDENTIFY, 772
- PCI\_RES\_SAVE\_CONFIGURATION, 772
- PCI\_SAVE\_CONFIGURATION, 772
- SERVICE\_FAULT\_MEMORY\_CLEAR, 772
- SERVICE\_ASSIGN\_FRAME\_ID, 772
- SERVICE\_ASSIGN\_FRAME\_ID\_RANGE, 772
- SERVICE\_ASSIGN\_NAD, 772
- SERVICE\_CONDITIONAL\_CHANGE\_NAD, 772
- SERVICE\_FAULT\_MEMORY\_READ, 773
- SERVICE\_IO\_CONTROL\_BY\_IDENTIFY, 773
- SERVICE\_READ\_BY\_IDENTIFY, 773
- SERVICE\_READ\_DATA\_BY\_IDENTIFY, 773
- SERVICE\_SAVE\_CONFIGURATION, 773
- SERVICE\_SESSION\_CONTROL, 773
- SERVICE\_WRITE\_DATA\_BY\_IDENTIFY, 773
- TL\_ACTION\_ID\_IGNORE, 777
- TL\_ACTION\_NONE, 777
- TL\_ERROR, 777
- TL\_HANDLER\_INTERLEAVE\_MODE, 777
- TL\_MAKE\_RES\_DATA, 777
- TL\_RECEIVE\_MESSAGE, 777
- TL\_RX\_COMPLETED, 777
- TL\_SLAVE\_GET\_ACTION, 777
- TL\_TIMEOUT\_SERVICE, 777
- TL\_TX\_COMPLETED, 777
- timerGetTimeIntervalCallbackArr, 783
- Low Power Inter-Integrated Circuit (LPI2C), 745
- Low Power Interrupt Timer (LPIT), 746
- Low Power Serial Peripheral Interface (LPSPi), 747
- Low Power Timer (LPTMR), 750
- Low Power Universal Asynchronous Receiver-Transmitter (LPUART), 751
- lowSideCtrl
  - uja113x\_mtpnv\_cfg\_t, 208
- lpi2c\_baud\_rate\_params\_t, 675
  - baudRate, 676
- lpi2c\_master\_state\_t, 676
- lpi2c\_master\_user\_config\_t, 673
  - baudRate, 674
  - callbackParam, 674
  - dmaChannel, 674
  - is10bitAddr, 674
  - masterCallback, 674
  - operatingMode, 674
  - slaveAddress, 674
  - transferType, 674
- lpi2c\_mode\_t
  - LPI2C Driver, 676
- lpi2c\_slave\_state\_t, 676
- lpi2c\_slave\_user\_config\_t, 674
  - callbackParam, 675
  - dmaChannel, 675
  - is10bitAddr, 675
  - operatingMode, 675
  - slaveAddress, 675
  - slaveCallback, 675
  - slaveListening, 675
  - transferType, 675
- lpi2c\_transfer\_type\_t
  - LPI2C Driver, 676
- lpit\_period\_units\_t
  - LPIT Driver, 692
- lpit\_timer\_modes\_t
  - LPIT Driver, 692
- lpit\_trigger\_source\_t

- LPIT Driver, [692](#)
- lpit\_user\_channel\_config\_t, [690](#)
  - chainChannel, [690](#)
  - enableReloadOnTrigger, [690](#)
  - enableStartOnTrigger, [690](#)
  - enableStopOnInterrupt, [691](#)
  - isInterruptEnabled, [691](#)
  - period, [691](#)
  - periodUnits, [691](#)
  - timerMode, [691](#)
  - triggerSelect, [691](#)
  - triggerSource, [691](#)
- lpit\_user\_config\_t, [690](#)
  - enableRunInDebug, [690](#)
  - enableRunInDoze, [690](#)
- lpoClockConfig
  - pmc\_config\_t, [297](#)
  - sim\_clock\_config\_t, [288](#)
- lpspi\_clock\_phase\_t
  - LPSPi Driver, [709](#)
- lpspi\_master\_config\_t, [703](#)
  - bitcount, [703](#)
  - bitsPerSec, [703](#)
  - callback, [704](#)
  - callbackParam, [704](#)
  - clkPhase, [704](#)
  - clkPolarity, [704](#)
  - isPcsContinuous, [704](#)
  - lpspiSrcClk, [704](#)
  - lsbFirst, [704](#)
  - pcsPolarity, [704](#)
  - rxDMACHannel, [704](#)
  - transferType, [704](#)
  - txDMACHannel, [704](#)
  - whichPcs, [705](#)
- lpspi\_sck\_polarity\_t
  - LPSPi Driver, [709](#)
- lpspi\_signal\_polarity\_t
  - LPSPi Driver, [709](#)
- lpspi\_slave\_config\_t, [707](#)
  - bitcount, [708](#)
  - callback, [708](#)
  - callbackParam, [708](#)
  - clkPhase, [708](#)
  - clkPolarity, [708](#)
  - lsbFirst, [708](#)
  - pcsPolarity, [708](#)
  - rxDMACHannel, [708](#)
  - transferType, [708](#)
  - txDMACHannel, [708](#)
  - whichPcs, [709](#)
- lpspi\_state\_t, [705](#)
  - bitsPerFrame, [705](#)
  - bytesPerFrame, [705](#)
  - callback, [705](#)
  - callbackParam, [706](#)
  - dummy, [706](#)
  - fifoSize, [706](#)
  - isBlocking, [706](#)
  - isPcsContinuous, [706](#)
  - isTransferInProgress, [706](#)
  - lpspiSemaphore, [706](#)
  - lpspiSrcClk, [706](#)
  - lsb, [706](#)
  - rxBuff, [706](#)
  - rxCount, [706](#)
  - rxDMACHannel, [707](#)
  - rxFrameCnt, [707](#)
  - status, [707](#)
  - transferType, [707](#)
  - txBuff, [707](#)
  - txCount, [707](#)
  - txDMACHannel, [707](#)
  - txFrameCnt, [707](#)
- lpspi\_transfer\_type
  - LPSPi Driver, [709](#)
- lpspi\_which\_pcs\_t
  - LPSPi Driver, [709](#)
- lpspiInstance
  - uja113x\_driver\_t, [147](#)
  - uja113x\_drv\_cfg\_t, [142](#)
- lpspiIntace
  - drv\_config\_t, [1083](#)
- lpspiSemaphore
  - lpspi\_state\_t, [706](#)
- lpspiSrcClk
  - lpspi\_master\_config\_t, [704](#)
  - lpspi\_state\_t, [706](#)
- lptmr\_clocksource\_t
  - LPTMR Driver, [723](#)
- lptmr\_config\_t, [722](#)
  - bypassPrescaler, [722](#)
  - clockSelect, [722](#)
  - compareValue, [722](#)
  - counterUnits, [722](#)
  - dmaRequest, [722](#)
  - freeRun, [722](#)
  - interruptEnable, [723](#)
  - pinPolarity, [723](#)
  - pinSelect, [723](#)
  - prescaler, [723](#)
  - workMode, [723](#)
- lptmr\_counter\_units\_t
  - LPTMR Driver, [723](#)
- lptmr\_pinpolarity\_t
  - LPTMR Driver, [723](#)
- lptmr\_pinselect\_t
  - LPTMR Driver, [724](#)
- lptmr\_prescaler\_t
  - LPTMR Driver, [724](#)
- lptmr\_workmode\_t
  - LPTMR Driver, [724](#)
- lpuart\_bit\_count\_per\_char\_t
  - LPUART Driver, [736](#)
- lpuart\_parity\_mode\_t
  - LPUART Driver, [736](#)

- lpuart\_state\_t, 732
  - bitCountPerChar, 733
  - isRxBlocking, 733
  - isRxBusy, 733
  - isTxBlocking, 733
  - isTxBusy, 733
  - receiveStatus, 733
  - rxBuff, 733
  - rxCallback, 733
  - rxCallbackParam, 734
  - rxComplete, 734
  - rxSize, 734
  - transferType, 734
  - transmitStatus, 734
  - txBuff, 734
  - txCallback, 734
  - txCallbackParam, 734
  - txComplete, 734
  - txSize, 734
- lpuart\_stop\_bit\_count\_t
  - LPUART Driver, 736
- lpuart\_transfer\_type\_t
  - LPUART Driver, 736
- lpuart\_user\_config\_t, 735
  - baudRate, 735
  - bitCountPerChar, 735
  - parityMode, 735
  - rxDMACChannel, 735
  - stopBitCount, 735
  - transferType, 735
  - txDMACChannel, 735
- lsb
  - lpspi\_state\_t, 706
- lsbFirst
  - lpspi\_master\_config\_t, 704
  - lpspi\_slave\_config\_t, 708
- MAKE\_PARITY
  - LIN Driver, 657
- MASTER
  - LIN Driver, 657
- MAX\_PERIOD\_COUNT
  - LPIT Driver, 691
- MAX\_PERIOD\_COUNT\_16\_BIT
  - LPIT Driver, 691
- MAX\_PERIOD\_COUNT\_IN\_DUAL\_16BIT\_MODE
  - LPIT Driver, 691
- MC\_ADDRESS
  - Register address map, 923
- MC\_NORMAL
  - 01. Mode handling, 159
- MC\_SLEEP
  - 01. Mode handling, 159
- MC\_STANDBY
  - 01. Mode handling, 159
- MEM0\_ADDRESS
  - Register address map, 923
- MEM1\_ADDRESS
  - Register address map, 923
- MEM2\_ADDRESS
  - Register address map, 923
- MEM3\_ADDRESS
  - Register address map, 924
- MINS\_IN\_A\_HOUR
  - RTC Driver, 899
- MPU Driver, 784
  - MPU\_DATA\_ACCESS\_IN\_SUPERVISOR\_MODE, 795
  - MPU\_DATA\_ACCESS\_IN\_USER\_MODE, 795
  - MPU\_DRV\_Deinit, 795
  - MPU\_DRV\_EnableRegion, 795
  - MPU\_DRV\_GetDefaultRegionConfig, 795
  - MPU\_DRV\_GetDetailErrorAccessInfo, 795
  - MPU\_DRV\_Init, 796
  - MPU\_DRV\_SetMasterAccessRights, 796
  - MPU\_DRV\_SetRegionAddr, 796
  - MPU\_DRV\_SetRegionConfig, 797
  - MPU\_ERR\_TYPE\_READ, 794
  - MPU\_ERR\_TYPE\_WRITE, 794
  - MPU\_INSTRUCTION\_ACCESS\_IN\_SUPERVISOR\_MODE, 795
  - MPU\_INSTRUCTION\_ACCESS\_IN\_USER\_MODE, 795
  - MPU\_NONE, 794
  - MPU\_R, 794
  - MPU\_RW, 794
  - MPU\_SUPERVISOR\_RW\_USER\_NONE, 794
  - MPU\_SUPERVISOR\_RW\_USER\_R, 794
  - MPU\_SUPERVISOR\_RW\_USER\_RW, 794
  - MPU\_SUPERVISOR\_RW\_USER\_RWX, 794
  - MPU\_SUPERVISOR\_RW\_USER\_RX, 794
  - MPU\_SUPERVISOR\_RW\_USER\_W, 794
  - MPU\_SUPERVISOR\_RW\_USER\_WX, 794
  - MPU\_SUPERVISOR\_RW\_USER\_X, 794
  - MPU\_SUPERVISOR\_RWX\_USER\_NONE, 793
  - MPU\_SUPERVISOR\_RWX\_USER\_R, 794
  - MPU\_SUPERVISOR\_RWX\_USER\_RW, 794
  - MPU\_SUPERVISOR\_RWX\_USER\_RWX, 794
  - MPU\_SUPERVISOR\_RWX\_USER\_RX, 794
  - MPU\_SUPERVISOR\_RWX\_USER\_W, 793
  - MPU\_SUPERVISOR\_RWX\_USER\_WX, 793
  - MPU\_SUPERVISOR\_RWX\_USER\_X, 793
  - MPU\_SUPERVISOR\_RX\_USER\_NONE, 794
  - MPU\_SUPERVISOR\_RX\_USER\_R, 794
  - MPU\_SUPERVISOR\_RX\_USER\_RW, 794
  - MPU\_SUPERVISOR\_RX\_USER\_RWX, 794
  - MPU\_SUPERVISOR\_RX\_USER\_RX, 794
  - MPU\_SUPERVISOR\_RX\_USER\_W, 794
  - MPU\_SUPERVISOR\_RX\_USER\_WX, 794
  - MPU\_SUPERVISOR\_RX\_USER\_X, 794
  - MPU\_SUPERVISOR\_USER\_NONE, 794
  - MPU\_SUPERVISOR\_USER\_R, 794
  - MPU\_SUPERVISOR\_USER\_RW, 794
  - MPU\_SUPERVISOR\_USER\_RWX, 794
  - MPU\_SUPERVISOR\_USER\_RX, 794
  - MPU\_SUPERVISOR\_USER\_W, 794
  - MPU\_SUPERVISOR\_USER\_WX, 794

- MPU\_SUPERVISOR\_USER\_X, [794](#)
- MPU\_W, [794](#)
- mpu\_access\_rights\_t, [791](#)
- mpu\_err\_access\_type\_t, [794](#)
- mpu\_err\_attributes\_t, [794](#)
- MPU\_PAL, [798](#)
- MPU\_Deinit, [806](#)
- MPU\_ERROR\_SUPERVISOR\_MODE\_DATA\_ACCESS, [806](#)
- MPU\_ERROR\_SUPERVISOR\_MODE\_INSTRUCTION\_ACCESS, [806](#)
- MPU\_ERROR\_TYPE\_READ, [805](#)
- MPU\_ERROR\_TYPE\_WRITE, [805](#)
- MPU\_ERROR\_USER\_MODE\_DATA\_ACCESS, [806](#)
- MPU\_ERROR\_USER\_MODE\_INSTRUCTION\_ACCESS, [806](#)
- MPU\_EnableRegion, [806](#)
- MPU\_GetDefaultRegionConfig, [807](#)
- MPU\_GetError, [807](#)
- MPU\_Init, [807](#)
- MPU\_UpdateRegion, [808](#)
- mpu\_access\_permission\_t, [803](#)
- mpu\_error\_access\_type\_t, [805](#)
- mpu\_error\_attributes\_t, [805](#)
- mpu\_inst\_type\_t, [806](#)
- MPU\_DATA\_ACCESS\_IN\_SUPERVISOR\_MODE
  - MPU Driver, [795](#)
- MPU\_DATA\_ACCESS\_IN\_USER\_MODE
  - MPU Driver, [795](#)
- MPU\_DRV\_Deinit
  - MPU Driver, [795](#)
- MPU\_DRV\_EnableRegion
  - MPU Driver, [795](#)
- MPU\_DRV\_GetDefaultRegionConfig
  - MPU Driver, [795](#)
- MPU\_DRV\_GetDetailErrorAccessInfo
  - MPU Driver, [795](#)
- MPU\_DRV\_Init
  - MPU Driver, [796](#)
- MPU\_DRV\_SetMasterAccessRights
  - MPU Driver, [796](#)
- MPU\_DRV\_SetRegionAddr
  - MPU Driver, [796](#)
- MPU\_DRV\_SetRegionConfig
  - MPU Driver, [797](#)
- MPU\_Deinit
  - MPU PAL, [806](#)
- MPU\_ERR\_TYPE\_READ
  - MPU Driver, [794](#)
- MPU\_ERR\_TYPE\_WRITE
  - MPU Driver, [794](#)
- MPU\_ERROR\_SUPERVISOR\_MODE\_DATA\_ACCESS
  - MPU PAL, [806](#)
- MPU\_ERROR\_SUPERVISOR\_MODE\_INSTRUCTION\_ACCESS
  - MPU PAL, [806](#)
- MPU\_ERROR\_TYPE\_READ
  - MPU PAL, [805](#)
- MPU\_ERROR\_TYPE\_WRITE
  - MPU PAL, [805](#)
- MPU\_ERROR\_USER\_MODE\_DATA\_ACCESS
  - MPU PAL, [806](#)
- MPU\_ERROR\_USER\_MODE\_INSTRUCTION\_ACCESS
  - MPU PAL, [806](#)
- MPU\_EnableRegion
  - MPU PAL, [806](#)
- MPU\_GetDefaultRegionConfig
  - MPU PAL, [807](#)
- MPU\_GetError
  - MPU PAL, [807](#)
- MPU\_INSTRUCTION\_ACCESS\_IN\_SUPERVISOR\_MODE
  - MPU Driver, [795](#)
- MPU\_INSTRUCTION\_ACCESS\_IN\_USER\_MODE
  - MPU Driver, [795](#)
- MPU\_Init
  - MPU PAL, [807](#)
- MPU\_NONE
  - MPU Driver, [794](#)
- MPU\_R
  - MPU Driver, [794](#)
- MPU\_RW
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RW\_USER\_NONE
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RW\_USER\_R
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RW\_USER\_RW
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RW\_USER\_RWX
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RW\_USER\_RX
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RW\_USER\_W
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RW\_USER\_WX
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RW\_USER\_X
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RWX\_USER\_NONE
  - MPU Driver, [793](#)
- MPU\_SUPERVISOR\_RWX\_USER\_R
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RWX\_USER\_RW
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RWX\_USER\_RWX
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RWX\_USER\_RX
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RWX\_USER\_W
  - MPU Driver, [793](#)
- MPU\_SUPERVISOR\_RWX\_USER\_WX
  - MPU Driver, [793](#)



- MPU\_SUPERVISOR\_RWX\_USER\_X
  - MPU Driver, [793](#)
- MPU\_SUPERVISOR\_RX\_USER\_NONE
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RX\_USER\_R
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RX\_USER\_RW
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RX\_USER\_RWX
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RX\_USER\_RX
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RX\_USER\_W
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RX\_USER\_WX
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_RX\_USER\_X
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_USER\_NONE
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_USER\_R
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_USER\_RW
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_USER\_RWX
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_USER\_RX
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_USER\_W
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_USER\_WX
  - MPU Driver, [794](#)
- MPU\_SUPERVISOR\_USER\_X
  - MPU Driver, [794](#)
- MPU\_UpdateRegion
  - MPU PAL, [808](#)
- MPU\_W
  - MPU Driver, [794](#)
- MSBCS\_ADDRESS
  - Register address map, [924](#)
- MTPNVCRCC\_ADDRESS
  - Register address map, [924](#)
- MTPNVS\_ADDRESS
  - Register address map, [924](#)
- MULTIPLY\_BY\_ONE
  - Clock Manager Driver, [305](#)
- MULTIPLY\_BY\_TWO
  - Clock Manager Driver, [305](#)
- mac
  - csec\_state\_t, [261](#)
- macLen
  - csec\_state\_t, [261](#)
- macWritten
  - csec\_state\_t, [262](#)
- mainChannelPolarity
  - ftm\_combined\_ch\_param\_t, [581](#)
- mainChannelSafeState
  - ftm\_combined\_ch\_param\_t, [581](#)
- mainS
  - sbc\_status\_group\_t, [1028](#)
- majorLoopChnLinkEnable
  - edma\_loop\_transfer\_config\_t, [379](#)
- majorLoopChnLinkNumber
  - edma\_loop\_transfer\_config\_t, [379](#)
- majorLoopIterationCount
  - edma\_loop\_transfer\_config\_t, [379](#)
- mask
  - sbc\_can\_conf\_t, [1019](#)
- maskRegSync
  - ftm\_pwm\_sync\_t, [523](#)
- master
  - mpu\_access\_err\_info\_t, [790](#)
  - mpu\_error\_info\_t, [801](#)
- master\_data\_buffer
  - lin\_master\_data\_t, [768](#)
- master\_ifc\_handle
  - lin\_protocol\_user\_config\_t, [766](#)
- masterAccRight
  - mpu\_region\_config\_t, [802](#)
  - mpu\_user\_config\_t, [791](#)
- masterCallback
  - lpi2c\_master\_user\_config\_t, [674](#)
- masterNum
  - mpu\_master\_access\_permission\_t, [802](#)
  - mpu\_master\_access\_right\_t, [790](#)
- max\_idle\_timeout\_cnt
  - lin\_protocol\_user\_config\_t, [767](#)
- max\_message\_length
  - lin\_protocol\_user\_config\_t, [767](#)
- max\_num\_mb
  - flexcan\_user\_config\_t, [451](#)
- maxBuffNum
  - can\_user\_config\_t, [350](#)
- maxCountValue
  - extension\_ftm\_for\_oc\_t, [839](#)
  - ftm\_output\_cmp\_param\_t, [567](#)
- maxLoadingPoint
  - ftm\_pwm\_sync\_t, [523](#)
- maxVal
  - ftm\_quad\_decode\_config\_t, [590](#)
- mb\_message
  - flexcan\_mb\_handle\_t, [447](#)
- mbSema
  - flexcan\_mb\_handle\_t, [447](#)
- mbs
  - FlexCANState, [448](#)
- mcSbcMode
  - uja113x\_init\_cfg\_t, [146](#)
- measurementResults
  - ftm\_state\_t, [522](#)
- measurementType
  - ftm\_input\_ch\_param\_t, [556](#)
- Memory Protection Unit (MPU), [809](#)
- Memory Protection Unit Peripheral Abstraction Layer (↔ MPU PAL), [811](#)
- memory0



- uja113x\_init\_cfg\_t, [146](#)
- memory1
  - uja113x\_init\_cfg\_t, [146](#)
- memory2
  - uja113x\_init\_cfg\_t, [146](#)
- memory3
  - uja113x\_init\_cfg\_t, [146](#)
- minLoadingPoint
  - ftm\_pwm\_sync\_t, [523](#)
- minorByteTransferCount
  - edma\_transfer\_config\_t, [381](#)
- minorLoopChnLinkEnable
  - edma\_loop\_transfer\_config\_t, [380](#)
- minorLoopChnLinkNumber
  - edma\_loop\_transfer\_config\_t, [380](#)
- minorLoopOffset
  - edma\_loop\_transfer\_config\_t, [380](#)
- minutes
  - rtc\_timedate\_t, [895](#)
- misoPin
  - extension\_flexio\_for\_spi\_t, [960](#)
  - flexio\_spi\_master\_user\_config\_t, [496](#)
  - flexio\_spi\_slave\_user\_config\_t, [498](#)
- mode
  - can\_user\_config\_t, [350](#)
  - cmp\_comparator\_t, [329](#)
  - ftm\_output\_cmp\_param\_t, [567](#)
  - ftm\_pwm\_param\_t, [582](#)
  - ftm\_quad\_decode\_config\_t, [590](#)
  - ftm\_timer\_param\_t, [563](#)
  - i2s\_user\_config\_t, [605](#)
  - sbc\_int\_config\_t, [1020](#)
- modeControl
  - sbc\_wtdog\_ctr\_t, [1013](#)
- module\_clk\_config\_t, [298](#)
  - div, [298](#)
  - gating, [298](#)
  - mul, [298](#)
  - source, [298](#)
- monitorMode
  - scg\_sosc\_config\_t, [290](#)
  - scg\_spill\_config\_t, [293](#)
- month
  - rtc\_timedate\_t, [895](#)
- mosiPin
  - extension\_flexio\_for\_spi\_t, [960](#)
  - flexio\_spi\_master\_user\_config\_t, [496](#)
  - flexio\_spi\_slave\_user\_config\_t, [498](#)
- mpu\_access\_err\_info\_t, [789](#)
  - accessCtr, [790](#)
  - accessType, [790](#)
  - addr, [790](#)
  - attributes, [790](#)
  - master, [790](#)
- mpu\_access\_permission\_t
  - MPU PAL, [803](#)
- mpu\_access\_rights\_t
  - MPU Driver, [791](#)
- mpu\_err\_access\_type\_t
  - MPU Driver, [794](#)
- mpu\_err\_attributes\_t
  - MPU Driver, [794](#)
- mpu\_error\_access\_type\_t
  - MPU PAL, [805](#)
- mpu\_error\_attributes\_t
  - MPU PAL, [805](#)
- mpu\_error\_info\_t, [801](#)
  - accessCtr, [801](#)
  - accessType, [801](#)
  - addr, [801](#)
  - attributes, [801](#)
  - master, [801](#)
  - overrun, [801](#)
  - processId, [801](#)
- mpu\_inst\_type\_t
  - MPU PAL, [806](#)
- mpu\_instance\_t, [1086](#)
  - instIdx, [1086](#)
  - instType, [1086](#)
- mpu\_master\_access\_permission\_t, [801](#)
  - accessRight, [802](#)
  - masterNum, [802](#)
- mpu\_master\_access\_right\_t, [790](#)
  - accessRight, [790](#)
  - masterNum, [790](#)
- mpu\_region\_config\_t, [802](#)
  - endAddr, [802](#)
  - extension, [802](#)
  - masterAccRight, [802](#)
  - processIdEnable, [802](#)
  - processIdMask, [803](#)
  - processIdentifier, [803](#)
  - startAddr, [803](#)
- mpu\_user\_config\_t, [790](#)
  - endAddr, [791](#)
  - masterAccRight, [791](#)
  - startAddr, [791](#)
- msg\_id\_type
  - flexcan\_data\_info\_t, [449](#)
- msgId
  - flexcan\_msgbuff\_t, [447](#)
- msgLen
  - csec\_state\_t, [262](#)
- mul
  - clock\_source\_config\_t, [299](#)
  - module\_clk\_config\_t, [298](#)
- mult
  - scg\_spill\_config\_t, [293](#)
- mux
  - cmp\_module\_t, [332](#)
  - pin\_settings\_config\_t, [858](#)
- N\_As\_timeout
  - lin\_node\_attribute\_t, [758](#)
- N\_Cr\_timeout
  - lin\_node\_attribute\_t, [758](#)
- NBYTES

- edma\_software\_tcd\_t, 383
- NEGATIVE
  - Common Transport Layer API, 320
- nMaxCountValue
  - ftm\_input\_param\_t, 557
- nNumChannels
  - ftm\_input\_param\_t, 557
  - ic\_config\_t, 616
  - oc\_config\_t, 839
- nNumCombinedPwmChannels
  - ftm\_pwm\_param\_t, 582
- nNumIndependentPwmChannels
  - ftm\_pwm\_param\_t, 582
- nNumOutputChannels
  - ftm\_output\_cmp\_param\_t, 568
- NO\_MODE
  - Clock Manager Driver, 305
- nTimer
  - uja113x\_timsc\_cfg\_t, 197
- NULL\_CALLBACK
  - Flash Memory (Flash), 425
- NUMBER\_OF\_TCLK\_INPUTS
  - Clock Manager Driver, 303
- NWP\_PERIOD\_1024
  - 02. Watchdog, 176
- NWP\_PERIOD\_128
  - 02. Watchdog, 176
- NWP\_PERIOD\_16
  - 02. Watchdog, 176
- NWP\_PERIOD\_256
  - 02. Watchdog, 176
- NWP\_PERIOD\_32
  - 02. Watchdog, 176
- NWP\_PERIOD\_4096
  - 02. Watchdog, 176
- NWP\_PERIOD\_64
  - 02. Watchdog, 176
- NWP\_PERIOD\_8
  - 02. Watchdog, 176
- negativeInputMux
  - cmp\_anmux\_t, 329
- negativePortMux
  - cmp\_anmux\_t, 329
- next\_transmit\_tick
  - lin\_protocol\_state\_t, 770
- nms
  - sbc\_main\_status\_t, 1022
- Node configuration, 816, 818
  - ld\_assign\_NAD, 818
  - ld\_assign\_NAD\_j2602, 816
  - ld\_assign\_frame\_id\_range, 818
  - ld\_check\_response, 820
  - ld\_check\_response\_j2602, 816
  - ld\_conditional\_change\_NAD, 820
  - ld\_is\_ready, 820
  - ld\_is\_ready\_j2602, 816
  - ld\_read\_configuration, 821
  - ld\_reconfig\_msg\_ID, 817
  - ld\_save\_configuration, 821
  - ld\_set\_configuration, 821
- Node identification, 823
  - ld\_read\_by\_id, 823
- nodeFunction
  - lin\_user\_config\_t, 654
- nomWtdPeriod
  - uja113x\_wtdc\_cfg\_t, 175
- nominalBitrate
  - can\_user\_config\_t, 350
- nominalPeriod
  - sbc\_wtdog\_ctr\_t, 1013
- nonSupervisorAccessEnable
  - rtc\_init\_config\_t, 896
- Notification, 824
- notifyType
  - clock\_notify\_struct\_t, 300
  - power\_manager\_notify\_struct\_t, 867
- num\_frame\_have\_esignal
  - lin\_node\_attribute\_t, 758
- num\_id\_filters
  - flexcan\_user\_config\_t, 451
- num\_of\_associated\_uncond\_frames
  - lin\_associate\_frame\_t, 760
- num\_of\_fault\_state\_signal
  - lin\_node\_attribute\_t, 758
- num\_of\_pdu
  - lin\_tl\_descriptor\_t, 764
- num\_of\_processed\_frame
  - lin\_protocol\_state\_t, 770
- num\_of\_schedules
  - lin\_protocol\_user\_config\_t, 767
- num\_slots
  - lin\_schedule\_t, 761
- numChan
  - timer\_config\_t, 997
- numChannels
  - adc\_group\_config\_t, 235
- numGroups
  - adc\_config\_t, 236
- numIdFilters
  - extension\_flexcan\_rx\_fifo\_t, 351
- numInOutMappingConfigs
  - trgmux\_user\_config\_t, 987
- numOfClassicPID
  - lin\_user\_config\_t, 654
- numOfRecordReqMaintain
  - Flash Memory (Flash), 435
- numSetsResultBuffer
  - adc\_group\_config\_t, 235
- number\_of\_configurable\_frames
  - lin\_protocol\_user\_config\_t, 767
- number\_support\_sid
  - lin\_node\_attribute\_t, 758
- numberOfPwmChannels
  - pwm\_global\_config\_t, 889
- numberOfRepeats
  - rtc\_alarm\_config\_t, 897

- nvmps
  - sbc\_mtpnv\_stat\_t, [1028](#)
- OC\_ABSOLUTE\_VALUE
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [840](#)
- OC\_CLEAR\_ON\_MATCH
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [839](#)
- OC\_DISABLE\_OUTPUT
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [839](#)
- OC\_Deinit
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [840](#)
- OC\_DisableNotification
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [840](#)
- OC\_EnableNotification
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [840](#)
- OC\_Init
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [840](#)
- OC\_RELATIVE\_VALUE
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [840](#)
- OC\_SET\_ON\_MATCH
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [839](#)
- OC\_SetCompareValue
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [841](#)
- OC\_SetOutputAction
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [841](#)
- OC\_SetOutputState
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [842](#)
- OC\_StartChannel
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [842](#)
- OC\_StopChannel
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [842](#)
- OC\_TOGGLE\_ON\_MATCH
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [839](#)
- OS Interface (OSIF), [825](#)
  - OSIF\_GetMilliseconds, [827](#)
  - OSIF\_MutexCreate, [827](#)
  - OSIF\_MutexDestroy, [829](#)
  - OSIF\_MutexLock, [829](#)
  - OSIF\_MutexUnlock, [829](#)
  - OSIF\_SemaCreate, [829](#)
  - OSIF\_SemaDestroy, [831](#)
  - OSIF\_SemaPost, [831](#)
  - OSIF\_SemaWait, [831](#)
  - OSIF\_TimeDelay, [831](#)
  - OSIF\_WAIT\_FOREVER, [827](#)
- OSIF\_GetMilliseconds
  - OS Interface (OSIF), [827](#)
- OSIF\_MutexCreate
  - OS Interface (OSIF), [827](#)
- OSIF\_MutexDestroy
  - OS Interface (OSIF), [829](#)
- OSIF\_MutexLock
  - OS Interface (OSIF), [829](#)
- OSIF\_MutexUnlock
  - OS Interface (OSIF), [829](#)
- OSIF\_SemaCreate
  - OS Interface (OSIF), [829](#)
- OSIF\_SemaDestroy
  - OS Interface (OSIF), [831](#)
- OSIF\_SemaPost
  - OS Interface (OSIF), [831](#)
- OSIF\_SemaWait
  - OS Interface (OSIF), [831](#)
- OSIF\_TimeDelay
  - OS Interface (OSIF), [831](#)
- OSIF\_WAIT\_FOREVER
  - OS Interface (OSIF), [827](#)
- OTWIE\_DISABLE
  - System interrupt enable configuration options (0x04), [984](#)
- OTWIE\_ENABLE
  - System interrupt enable configuration options (0x04), [984](#)
- OVERRUN
  - Common Core API., [316](#)
- oc\_config\_t, [838](#)
  - extension, [838](#)
  - nNumChannels, [839](#)
  - outputChConfig, [839](#)
- oc\_instance\_t, [1086](#)
  - instIdx, [1087](#)
  - instType, [1087](#)
- oc\_option\_mode\_t
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [839](#)
- oc\_option\_update\_t
  - Output Compare - Peripheral Abstraction Layer (↔ OC PAL), [839](#)
- oc\_output\_ch\_param\_t, [837](#)
  - chMode, [838](#)
  - channelCallbackParams, [838](#)
  - channelCallbacks, [838](#)
  - channelExtension, [838](#)
  - comparedValue, [838](#)
  - hwChannelId, [838](#)
- oc\_pal\_state\_t, [1087](#)
- opMode
  - wdg\_config\_t, [1063](#)
  - wdog\_user\_config\_t, [1071](#)
- operatingMode
  - i2c\_master\_t, [625](#)
  - i2c\_slave\_t, [626](#)

- lpi2c\_master\_user\_config\_t, [674](#)
- lpi2c\_slave\_user\_config\_t, [675](#)
- otw
  - sbc\_sys\_evnt\_stat\_t, [1025](#)
- otws
  - sbc\_main\_status\_t, [1022](#)
- outRegSync
  - ftm\_pwm\_sync\_t, [524](#)
- Output Compare - Peripheral Abstraction Layer (OC P↔AL), [834](#)
  - OC\_ABSOLUTE\_VALUE, [840](#)
  - OC\_CLEAR\_ON\_MATCH, [839](#)
  - OC\_DISABLE\_OUTPUT, [839](#)
  - OC\_Deinit, [840](#)
  - OC\_DisableNotification, [840](#)
  - OC\_EnableNotification, [840](#)
  - OC\_Init, [840](#)
  - OC\_RELATIVE\_VALUE, [840](#)
  - OC\_SET\_ON\_MATCH, [839](#)
  - OC\_SetCompareValue, [841](#)
  - OC\_SetOutputAction, [841](#)
  - OC\_SetOutputState, [842](#)
  - OC\_StartChannel, [842](#)
  - OC\_StopChannel, [842](#)
  - OC\_TOGGLE\_ON\_MATCH, [839](#)
  - oc\_option\_mode\_t, [839](#)
  - oc\_option\_update\_t, [839](#)
- outputBuff
  - csec\_state\_t, [262](#)
- outputChConfig
  - oc\_config\_t, [839](#)
- outputChannelConfig
  - ftm\_output\_cmp\_param\_t, [568](#)
- outputDiv1
  - clock\_source\_config\_t, [299](#)
- outputDiv2
  - clock\_source\_config\_t, [300](#)
- outputInterruptTrigger
  - cmp\_comparator\_t, [329](#)
- outputSelect
  - cmp\_comparator\_t, [329](#)
- overflowDirection
  - ftm\_quad\_decoder\_state\_t, [591](#)
- overflowFlag
  - ftm\_quad\_decoder\_state\_t, [591](#)
- overflowIntEnable
  - rtc\_interrupt\_config\_t, [898](#)
- overrun
  - lin\_word\_status\_str\_t, [756](#)
  - mpu\_error\_info\_t, [801](#)
- overrun\_flg
  - lin\_protocol\_state\_t, [770](#)
- owte
  - sbc\_sys\_evnt\_t, [1016](#)
- P2\_min
  - lin\_node\_attribute\_t, [758](#)
- PCI\_RES\_ASSIGN\_FRAME\_ID\_RANGE
  - Low level API, [772](#)
- PCI\_RES\_READ\_BY\_IDENTIFY
  - Low level API, [772](#)
- PCI\_RES\_SAVE\_CONFIGURATION
  - Low level API, [772](#)
- PCI\_SAVE\_CONFIGURATION
  - Low level API, [772](#)
- PDB Driver, [845](#)
  - PDB\_CLK\_PREDIV\_BY\_1, [850](#)
  - PDB\_CLK\_PREDIV\_BY\_128, [851](#)
  - PDB\_CLK\_PREDIV\_BY\_16, [850](#)
  - PDB\_CLK\_PREDIV\_BY\_2, [850](#)
  - PDB\_CLK\_PREDIV\_BY\_32, [850](#)
  - PDB\_CLK\_PREDIV\_BY\_4, [850](#)
  - PDB\_CLK\_PREDIV\_BY\_64, [850](#)
  - PDB\_CLK\_PREDIV\_BY\_8, [850](#)
  - PDB\_CLK\_PREMULT\_FACT\_AS\_1, [851](#)
  - PDB\_CLK\_PREMULT\_FACT\_AS\_10, [851](#)
  - PDB\_CLK\_PREMULT\_FACT\_AS\_20, [851](#)
  - PDB\_CLK\_PREMULT\_FACT\_AS\_40, [851](#)
  - PDB\_DRV\_ClearAdcPreTriggerFlags, [851](#)
  - PDB\_DRV\_ClearAdcPreTriggerSeqErrFlags, [852](#)
  - PDB\_DRV\_ClearTimerIntFlag, [852](#)
  - PDB\_DRV\_ConfigAdcPreTrigger, [852](#)
  - PDB\_DRV\_Deinit, [852](#)
  - PDB\_DRV\_Disable, [852](#)
  - PDB\_DRV\_Enable, [853](#)
  - PDB\_DRV\_GetAdcPreTriggerFlags, [853](#)
  - PDB\_DRV\_GetAdcPreTriggerSeqErrFlags, [853](#)
  - PDB\_DRV\_GetDefaultConfig, [853](#)
  - PDB\_DRV\_GetTimerIntFlag, [854](#)
  - PDB\_DRV\_GetTimerValue, [854](#)
  - PDB\_DRV\_Init, [854](#)
  - PDB\_DRV\_LoadValuesCmd, [854](#)
  - PDB\_DRV\_SetAdcPreTriggerDelayValue, [855](#)
  - PDB\_DRV\_SetCmpPulseOutDelayForHigh, [855](#)
  - PDB\_DRV\_SetCmpPulseOutDelayForLow, [855](#)
  - PDB\_DRV\_SetCmpPulseOutEnable, [855](#)
  - PDB\_DRV\_SetTimerModulusValue, [856](#)
  - PDB\_DRV\_SetValueForTimerInterrupt, [856](#)
  - PDB\_DRV\_SoftTriggerCmd, [856](#)
  - PDB\_LOAD\_VAL\_AT\_MODULO\_COUNTER, [851](#)
  - PDB\_LOAD\_VAL\_AT\_MODULO\_COUNTER\_O↔R\_NEXT\_TRIGGER, [851](#)
  - PDB\_LOAD\_VAL\_AT\_NEXT\_TRIGGER, [851](#)
  - PDB\_LOAD\_VAL\_IMMEDIATELY, [851](#)
  - PDB\_SOFTWARE\_TRIGGER, [851](#)
  - PDB\_TRIGGER\_IN0, [851](#)
  - pdb\_clk\_prescaler\_div\_t, [850](#)
  - pdb\_clk\_prescaler\_mult\_factor\_t, [851](#)
  - pdb\_load\_value\_mode\_t, [851](#)
  - pdb\_trigger\_src\_t, [851](#)
- PDB\_CLK\_PREDIV\_BY\_1
  - PDB Driver, [850](#)
- PDB\_CLK\_PREDIV\_BY\_128
  - PDB Driver, [851](#)
- PDB\_CLK\_PREDIV\_BY\_16
  - PDB Driver, [850](#)
- PDB\_CLK\_PREDIV\_BY\_2

- PDB Driver, [850](#)
- PDB\_CLK\_PREDIV\_BY\_32
  - PDB Driver, [850](#)
- PDB\_CLK\_PREDIV\_BY\_4
  - PDB Driver, [850](#)
- PDB\_CLK\_PREDIV\_BY\_64
  - PDB Driver, [850](#)
- PDB\_CLK\_PREDIV\_BY\_8
  - PDB Driver, [850](#)
- PDB\_CLK\_PREMULT\_FACT\_AS\_1
  - PDB Driver, [851](#)
- PDB\_CLK\_PREMULT\_FACT\_AS\_10
  - PDB Driver, [851](#)
- PDB\_CLK\_PREMULT\_FACT\_AS\_20
  - PDB Driver, [851](#)
- PDB\_CLK\_PREMULT\_FACT\_AS\_40
  - PDB Driver, [851](#)
- PDB\_DRV\_ClearAdcPreTriggerFlags
  - PDB Driver, [851](#)
- PDB\_DRV\_ClearAdcPreTriggerSeqErrFlags
  - PDB Driver, [852](#)
- PDB\_DRV\_ClearTimerIntFlag
  - PDB Driver, [852](#)
- PDB\_DRV\_ConfigAdcPreTrigger
  - PDB Driver, [852](#)
- PDB\_DRV\_Deinit
  - PDB Driver, [852](#)
- PDB\_DRV\_Disable
  - PDB Driver, [852](#)
- PDB\_DRV\_Enable
  - PDB Driver, [853](#)
- PDB\_DRV\_GetAdcPreTriggerFlags
  - PDB Driver, [853](#)
- PDB\_DRV\_GetAdcPreTriggerSeqErrFlags
  - PDB Driver, [853](#)
- PDB\_DRV\_GetDefaultConfig
  - PDB Driver, [853](#)
- PDB\_DRV\_GetTimerIntFlag
  - PDB Driver, [854](#)
- PDB\_DRV\_GetTimerValue
  - PDB Driver, [854](#)
- PDB\_DRV\_Init
  - PDB Driver, [854](#)
- PDB\_DRV\_LoadValuesCmd
  - PDB Driver, [854](#)
- PDB\_DRV\_SetAdcPreTriggerDelayValue
  - PDB Driver, [855](#)
- PDB\_DRV\_SetCmpPulseOutDelayForHigh
  - PDB Driver, [855](#)
- PDB\_DRV\_SetCmpPulseOutDelayForLow
  - PDB Driver, [855](#)
- PDB\_DRV\_SetCmpPulseOutEnable
  - PDB Driver, [855](#)
- PDB\_DRV\_SetTimerModulusValue
  - PDB Driver, [856](#)
- PDB\_DRV\_SetValueForTimerInterrupt
  - PDB Driver, [856](#)
- PDB\_DRV\_SoftTriggerCmd
  - PDB Driver, [856](#)
- PDB\_LOAD\_VAL\_AT\_MODULO\_COUNTER
  - PDB Driver, [851](#)
- PDB\_LOAD\_VAL\_AT\_MODULO\_COUNTER\_OR\_NEXT\_TRIGGER
  - PDB Driver, [851](#)
- PDB\_LOAD\_VAL\_NEXT\_TRIGGER
  - PDB Driver, [851](#)
- PDB\_LOAD\_VAL\_IMMEDIATELY
  - PDB Driver, [851](#)
- PDB\_SOFTWARE\_TRIGGER
  - PDB Driver, [851](#)
- PDB\_TRIGGER\_IN0
  - PDB Driver, [851](#)
- PFlashBase
  - Flash Memory (Flash), [435](#)
- PFlashSize
  - Flash Memory (Flash), [435](#)
- PINS Driver, [857](#)
  - GPIO\_INPUT\_DIRECTION, [858](#)
  - GPIO\_OUTPUT\_DIRECTION, [858](#)
  - GPIO\_UNSPECIFIED\_DIRECTION, [858](#)
  - PINS\_DRV\_ClearPins, [859](#)
  - PINS\_DRV\_GetPinsOutput, [859](#)
  - PINS\_DRV\_Init, [859](#)
  - PINS\_DRV\_ReadPins, [859](#)
  - PINS\_DRV\_SetPins, [860](#)
  - PINS\_DRV\_TogglePins, [860](#)
  - PINS\_DRV\_WritePin, [860](#)
  - PINS\_DRV\_WritePins, [861](#)
  - pins\_level\_type\_t, [858](#)
  - port\_data\_direction\_t, [858](#)
- PINS\_DRV\_ClearPins
  - PINS Driver, [859](#)
- PINS\_DRV\_GetPinsOutput
  - PINS Driver, [859](#)
- PINS\_DRV\_Init
  - PINS Driver, [859](#)
- PINS\_DRV\_ReadPins
  - PINS Driver, [859](#)
- PINS\_DRV\_SetPins
  - PINS Driver, [860](#)
- PINS\_DRV\_TogglePins
  - PINS Driver, [860](#)
- PINS\_DRV\_WritePin
  - PINS Driver, [860](#)
- PINS\_DRV\_WritePins
  - PINS Driver, [861](#)
- POSITIVE
  - Common Transport Layer API, [320](#)
- POWER\_MANAGER\_CALLBACK\_AFTER
  - Power Manager, [869](#)
- POWER\_MANAGER\_CALLBACK\_BEFORE
  - Power Manager, [869](#)
- POWER\_MANAGER\_CALLBACK\_BEFORE\_AFTER
  - Power Manager, [869](#)
- POWER\_MANAGER\_MAX
  - Power\_s32k1xx, [879](#)

- POWER\_MANAGER\_NOTIFY\_AFTER
  - Power Manager, [870](#)
- POWER\_MANAGER\_NOTIFY\_BEFORE
  - Power Manager, [870](#)
- POWER\_MANAGER\_NOTIFY\_RECOVER
  - Power Manager, [870](#)
- POWER\_MANAGER\_POLICY\_AGREEMENT
  - Power Manager, [870](#)
- POWER\_MANAGER\_POLICY\_FORCIBLE
  - Power Manager, [870](#)
- POWER\_MANAGER\_RUN
  - Power\_s32k1xx, [879](#)
- POWER\_MANAGER\_VLPR
  - Power\_s32k1xx, [879](#)
- POWER\_MANAGER\_VLPS
  - Power\_s32k1xx, [879](#)
- POWER\_SYS\_Deinit
  - Power Manager, [870](#)
- POWER\_SYS\_DoDeinit
  - Power\_s32k1xx, [881](#)
- POWER\_SYS\_DoGetDefaultConfig
  - Power\_s32k1xx, [881](#)
- POWER\_SYS\_DoInit
  - Power\_s32k1xx, [881](#)
- POWER\_SYS\_DoSetMode
  - Power\_s32k1xx, [881](#)
- POWER\_SYS\_GetCurrentMode
  - Power Manager, [870](#)
- POWER\_SYS\_GetDefaultConfig
  - Power Manager, [870](#)
- POWER\_SYS\_GetErrorCallback
  - Power Manager, [870](#)
- POWER\_SYS\_GetErrorCallbackIndex
  - Power Manager, [871](#)
- POWER\_SYS\_GetLastMode
  - Power Manager, [871](#)
- POWER\_SYS\_GetLastModeConfig
  - Power Manager, [871](#)
- POWER\_SYS\_GetResetSrcStatusCmd
  - Power\_s32k1xx, [881](#)
- POWER\_SYS\_Init
  - Power Manager, [872](#)
- POWER\_SYS\_SetMode
  - Power Manager, [872](#)
- PWM\_ACTIVE\_HIGH
  - Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), [889](#)
- PWM\_ACTIVE\_LOW
  - Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), [889](#)
- PWM\_CENTER\_ALIGNED
  - Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), [889](#)
- PWM\_DUPLICATED
  - Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), [889](#)
- PWM\_Deinit
  - Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), [889](#)
- PWM\_EDGE\_ALIGNED
  - Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), [889](#)
- PWM\_INVERTED
  - Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), [889](#)
- PWM\_Init
  - Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), [890](#)
- PWM\_OverwriteOutputChannels
  - Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), [890](#)
- PWM\_UpdateDuty
  - Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), [890](#)
- PWM\_UpdatePeriod
  - Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), [890](#)
- parameter
  - edma\_chn\_state\_t, [377](#)
- parityMode
  - lpuart\_user\_config\_t, [735](#)
  - uart\_user\_config\_t, [1052](#)
- partNetDataMask
  - uja113x\_cpnc\_cfg\_t, [153](#)
- partSize
  - csec\_state\_t, [262](#)
- payloadSize
  - can\_user\_config\_t, [350](#)
- pcc\_config\_t, [296](#)
  - count, [296](#)
  - peripheralClocks, [296](#)
- pccConfig
  - clock\_manager\_user\_config\_t, [298](#)
- pcsPolarity
  - lpspi\_master\_config\_t, [704](#)
  - lpspi\_slave\_config\_t, [708](#)
- pdb\_adc\_pretrigger\_config\_t, [850](#)
  - adcPreTriggerIdx, [850](#)
  - preTriggerBackToBackEnable, [850](#)
  - preTriggerEnable, [850](#)
  - preTriggerOutputEnable, [850](#)
- pdb\_clk\_prescaler\_div\_t
  - PDB Driver, [850](#)
- pdb\_clk\_prescaler\_mult\_factor\_t
  - PDB Driver, [851](#)
- pdb\_load\_value\_mode\_t
  - PDB Driver, [851](#)
- pdb\_timer\_config\_t, [849](#)
  - clkPreDiv, [849](#)
  - clkPreMultFactor, [849](#)
  - continuousModeEnable, [849](#)
  - dmaEnable, [849](#)
  - intEnable, [849](#)
  - loadValueMode, [849](#)
  - seqErrIntEnable, [849](#)



- triggerInput, [849](#)
- pdb\_trigger\_src\_t
  - PDB Driver, [851](#)
- pdbPrescaler
  - extension\_adc\_s32k1xx\_t, [236](#)
- pdc
  - sbc\_regulator\_t, [1015](#)
- peClkSrc
  - can\_user\_config\_t, [350](#)
- percentWindow
  - wdg\_config\_t, [1063](#)
- period
  - lpit\_user\_channel\_config\_t, [691](#)
  - pwm\_channel\_t, [888](#)
- periodUnits
  - lpit\_user\_channel\_config\_t, [691](#)
- Peripheral access layer for S32K142, [862](#)
- peripheral\_clock\_config\_t, [295](#)
  - clkGate, [296](#)
  - clkSrc, [296](#)
  - clockName, [296](#)
  - divider, [296](#)
  - frac, [296](#)
- peripheral\_clock\_divider\_t
  - Clock Manager Driver, [304](#)
- peripheral\_clock\_frac\_t
  - Clock Manager Driver, [304](#)
- peripheral\_clock\_source\_t
  - Clock Manager Driver, [303](#)
- peripheralClocks
  - pcc\_config\_t, [296](#)
- peripheralFeaturesList
  - Clock Manager Driver, [315](#)
- phaseAConfig
  - ftm\_quad\_decode\_config\_t, [591](#)
- phaseBConfig
  - ftm\_quad\_decode\_config\_t, [591](#)
- phaseFilterVal
  - ftm\_phase\_params\_t, [590](#)
- phaseInputFilter
  - ftm\_phase\_params\_t, [590](#)
- phasePolarity
  - ftm\_phase\_params\_t, [590](#)
- phaseSeg1
  - can\_time\_segment\_t, [348](#)
  - flexcan\_time\_segment\_t, [450](#)
- phaseSeg2
  - can\_time\_segment\_t, [348](#)
  - flexcan\_time\_segment\_t, [450](#)
- pin\_settings\_config\_t, [857](#)
  - direction, [858](#)
  - gpioBase, [858](#)
  - initValue, [858](#)
  - mux, [858](#)
  - pinPortIdx, [858](#)
- pinPolarity
  - lptmr\_config\_t, [723](#)
- pinPortIdx
  - pin\_settings\_config\_t, [858](#)
- pinSelect
  - lptmr\_config\_t, [723](#)
- pinState
  - cmp\_comparator\_t, [329](#)
- Pins Driver (PINS), [863](#)
- pins\_level\_type\_t
  - PINS Driver, [858](#)
- platGateConfig
  - sim\_clock\_config\_t, [288](#)
- pmc\_config\_t, [297](#)
  - lpoClockConfig, [297](#)
- pmc\_lpo\_clock\_config\_t, [296](#)
  - enable, [297](#)
  - initialize, [297](#)
  - trimValue, [297](#)
- pmcConfig
  - clock\_manager\_user\_config\_t, [298](#)
- pncok
  - sbc\_can\_ctr\_t, [1017](#)
- pndm
  - sbc\_frame\_t, [1018](#)
- pnfde
  - sbc\_trans\_evnt\_stat\_t, [1026](#)
- po
  - sbc\_sys\_evnt\_stat\_t, [1025](#)
- polarity
  - ftm\_independent\_ch\_param\_t, [579](#)
  - pwm\_channel\_t, [888](#)
- policy
  - clock\_notify\_struct\_t, [300](#)
  - power\_manager\_notify\_struct\_t, [867](#)
- poly
  - crc\_spec\_t, [208](#)
- port\_data\_direction\_t
  - PINS Driver, [858](#)
- positiveInputMux
  - cmp\_anmux\_t, [330](#)
- positivePortMux
  - cmp\_anmux\_t, [330](#)
- Power Manager, [865](#)
  - gPowerManagerState, [874](#)
  - POWER\_MANAGER\_CALLBACK\_AFTER, [869](#)
  - POWER\_MANAGER\_CALLBACK\_BEFORE, [869](#)
  - POWER\_MANAGER\_CALLBACK\_BEFORE\_AFTER, [869](#)
  - POWER\_MANAGER\_NOTIFY\_AFTER, [870](#)
  - POWER\_MANAGER\_NOTIFY\_BEFORE, [870](#)
  - POWER\_MANAGER\_NOTIFY\_RECOVER, [870](#)
  - POWER\_MANAGER\_POLICY\_AGREEMENT, [870](#)
  - POWER\_MANAGER\_POLICY\_FORCIBLE, [870](#)
  - POWER\_SYS\_Deinit, [870](#)
  - POWER\_SYS\_GetCurrentMode, [870](#)
  - POWER\_SYS\_GetDefaultConfig, [870](#)
  - POWER\_SYS\_GetErrorCallback, [870](#)
  - POWER\_SYS\_GetErrorCallbackIndex, [871](#)
  - POWER\_SYS\_GetLastMode, [871](#)

- POWER\_SYS\_GetLastModeConfig, 871
- POWER\_SYS\_Init, 872
- POWER\_SYS\_SetMode, 872
- power\_manager\_callback\_data\_t, 868
- power\_manager\_callback\_t, 869
- power\_manager\_callback\_type\_t, 869
- power\_manager\_notify\_t, 869
- power\_manager\_policy\_t, 870
- Power Manager Driver, 875
- power\_manager\_callback\_data\_t
  - Power Manager, 868
- power\_manager\_callback\_t
  - Power Manager, 869
- power\_manager\_callback\_type\_t
  - Power Manager, 869
- power\_manager\_callback\_user\_config\_t, 867
  - callbackData, 867
  - callbackFunction, 867
  - callbackType, 867
- power\_manager\_modes\_t
  - Power\_s32k1xx, 879
- power\_manager\_notify\_struct\_t, 866
  - notifyType, 867
  - policy, 867
  - targetPowerConfigIndex, 867
  - targetPowerConfigPtr, 867
- power\_manager\_notify\_t
  - Power Manager, 869
- power\_manager\_policy\_t
  - Power Manager, 870
- power\_manager\_state\_t, 867
  - configs, 868
  - configsNumber, 868
  - currentConfig, 868
  - errorCallbackIndex, 868
  - staticCallbacks, 868
  - staticCallbacksNumber, 868
- power\_manager\_user\_config\_t, 878
  - powerMode, 878
  - sleepOnExitValue, 878
- power\_mode\_stat\_t
  - Power\_s32k1xx, 879
- Power\_s32k1xx, 877
  - POWER\_MANAGER\_MAX, 879
  - POWER\_MANAGER\_RUN, 879
  - POWER\_MANAGER\_VLPR, 879
  - POWER\_MANAGER\_VLPS, 879
  - POWER\_SYS\_DoDeinit, 881
  - POWER\_SYS\_DoGetDefaultConfig, 881
  - POWER\_SYS\_DoInit, 881
  - POWER\_SYS\_DoSetMode, 881
  - POWER\_SYS\_GetResetSrcStatusCmd, 881
  - power\_manager\_modes\_t, 879
  - power\_mode\_stat\_t, 879
  - RCM\_CORE\_LOCKUP, 880
  - RCM\_EXTERNAL\_PIN, 880
  - RCM\_LOSS\_OF\_CLK, 880
  - RCM\_LOSS\_OF\_LOCK, 880
  - RCM\_LOW\_VOLT\_DETECT, 880
  - RCM\_POWER\_ON, 880
  - RCM\_SJTAG, 880
  - RCM\_SMDM\_AP, 880
  - RCM\_SOFTWARE, 880
  - RCM\_SRC\_NAME\_MAX, 880
  - RCM\_STOP\_MODE\_ACK\_ERR, 880
  - RCM\_WATCH\_DOG, 880
  - rcm\_source\_names\_t, 879
  - SMC\_HSRUN, 880
  - SMC\_RESERVED\_RUN, 880
  - SMC\_RESERVED\_STOP1, 880
  - SMC\_RUN, 880
  - SMC\_STOP, 880
  - SMC\_STOP1, 880
  - SMC\_STOP2, 880
  - SMC\_STOP\_RESERVED, 880
  - SMC\_VLPR, 880
  - SMC\_VLPS, 880
  - STAT\_HSRUN, 879
  - STAT\_INVALID, 879
  - STAT\_RUN, 879
  - STAT\_STOP, 879
  - STAT\_VLPR, 879
  - STAT\_VLPS, 879
  - STAT\_VLPW, 879
  - smc\_run\_mode\_t, 880
  - smc\_stop\_mode\_t, 880
  - smc\_stop\_option\_t, 880
- powerMode
  - cmp\_comparator\_t, 329
  - power\_manager\_user\_config\_t, 878
- powerModeName
  - smc\_power\_mode\_config\_t, 878
- preDivider
  - can\_time\_segment\_t, 348
  - flexcan\_time\_segment\_t, 450
- preTriggerBackToBackEnable
  - pdb\_adc\_pretrigger\_config\_t, 850
- preTriggerEnable
  - pdb\_adc\_pretrigger\_config\_t, 850
- preTriggerOutputEnable
  - pdb\_adc\_pretrigger\_config\_t, 850
- prediv
  - scg\_spill\_config\_t, 293
- prescaler
  - ewm\_init\_config\_t, 408
  - extension\_ftm\_for\_timer\_t, 998
  - extension\_lptmr\_for\_timer\_t, 998
  - lptmr\_config\_t, 723
  - pwm\_ftm\_timebase\_t, 887
- prescalerEnable
  - wdg\_config\_t, 1063
  - wdog\_user\_config\_t, 1072
- prescalerValue
  - extension\_ewm\_for\_wdg\_t, 1062
- pretriggerSel
  - adc\_converter\_config\_t, 217



- previous\_schedule\_id
  - lin\_master\_data\_t, 768
- processId
  - mpu\_error\_info\_t, 801
- processIdEnable
  - mpu\_region\_config\_t, 802
- processIdMask
  - mpu\_region\_config\_t, 803
- processIdentifier
  - mpu\_region\_config\_t, 803
- product\_id
  - lin\_node\_attribute\_t, 758
- product\_id\_ptr
  - lin\_tl\_descriptor\_t, 764
- programedState
  - cmp\_trigger\_mode\_t, 331
- Programmable Delay Block (PDB), 883
- propSeg
  - can\_time\_segment\_t, 348
  - flexcan\_time\_segment\_t, 450
- protocol\_version
  - lin\_protocol\_user\_config\_t, 767
- ptr\_sch\_data\_ptr
  - lin\_schedule\_t, 761
- Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), 884
  - PWM\_ACTIVE\_HIGH, 889
  - PWM\_ACTIVE\_LOW, 889
  - PWM\_CENTER\_ALIGNED, 889
  - PWM\_DUPLICATED, 889
  - PWM\_Deinit, 889
  - PWM\_EDGE\_ALIGNED, 889
  - PWM\_INVERTED, 889
  - PWM\_Init, 890
  - PWM\_OverwriteOutputChannels, 890
  - PWM\_UpdateDuty, 890
  - PWM\_UpdatePeriod, 890
  - pwm\_channel\_type\_t, 889
  - pwm\_complementary\_mode\_t, 889
  - pwm\_polarity\_t, 889
- pwm\_channel\_t, 887
  - channel, 887
  - channelType, 887
  - complementaryChannelPolarity, 888
  - deadtime, 888
  - duty, 888
  - enableComplementaryChannel, 888
  - insertDeadtime, 888
  - period, 888
  - polarity, 888
  - timebase, 888
- pwm\_channel\_type\_t
  - Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), 889
- pwm\_complementary\_mode\_t
  - Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), 889
- pwm\_ftm\_timebase\_t, 887
  - deadtimePrescaler, 887
  - prescaler, 887
  - sourceClock, 887
- pwm\_global\_config\_t, 888
  - numberOfPwmChannels, 889
  - pwmChannels, 889
- pwm\_instance\_t, 1087
  - instIdx, 1087
  - instType, 1088
- pwm\_polarity\_t
  - Pulse-width modulation - Peripheral Abstraction Layer (PWM PAL), 889
- pwmChannels
  - pwm\_global\_config\_t, 889
- pwmCombinedChannelConfig
  - ftm\_pwm\_param\_t, 582
- pwmFaultInterrupt
  - ftm\_pwm\_fault\_param\_t, 579
- pwmIndependentChannelConfig
  - ftm\_pwm\_param\_t, 582
- pwmOutputStateOnFault
  - ftm\_pwm\_fault\_param\_t, 579
- pwr\_modes\_t
  - Clock Manager Driver, 305
- qspiRefClkGating
  - sim\_clock\_config\_t, 288
- queue\_current\_size
  - lin\_transport\_layer\_queue\_t, 762
- queue\_header
  - lin\_transport\_layer\_queue\_t, 762
- queue\_max\_size
  - lin\_transport\_layer\_queue\_t, 762
- queue\_status
  - lin\_transport\_layer\_queue\_t, 762
- queue\_tail
  - lin\_transport\_layer\_queue\_t, 762
- RC\_ADDRESS
  - Register address map, 924
- RCM\_CORE\_LOCKUP
  - Power\_s32k1xx, 880
- RCM\_EXTERNAL\_PIN
  - Power\_s32k1xx, 880
- RCM\_LOSS\_OF\_CLK
  - Power\_s32k1xx, 880
- RCM\_LOSS\_OF\_LOCK
  - Power\_s32k1xx, 880
- RCM\_LOW\_VOLT\_DETECT
  - Power\_s32k1xx, 880
- RCM\_POWER\_ON
  - Power\_s32k1xx, 880
- RCM\_SJTAG
  - Power\_s32k1xx, 880
- RCM\_SMDM\_AP
  - Power\_s32k1xx, 880
- RCM\_SOFTWARE
  - Power\_s32k1xx, 880
- RCM\_SRC\_NAME\_MAX

- Power\_s32k1xx, [880](#)
- RCM\_STOP\_MODE\_ACK\_ERR
  - Power\_s32k1xx, [880](#)
- RCM\_WATCH\_DOG
  - Power\_s32k1xx, [880](#)
- READ\_ON\_EVEN\_EDGE
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [960](#)
- READ\_ON\_ODD\_EDGE
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [960](#)
- RECEIVING
  - Common Transport Layer API, [320](#)
- RES\_NEGATIVE
  - Common Transport Layer API, [320](#)
- RES\_POSITIVE
  - Common Transport Layer API, [320](#)
- RESUME\_WAIT\_CNT
  - Flash Memory (Flash), [425](#)
- rJumpwidth
  - can\_time\_segment\_t, [348](#)
  - flexcan\_time\_segment\_t, [450](#)
- RLC\_PULSE\_WIDTH\_01\_MS
  - Start-up control configuration options (0x73), [971](#)
- RLC\_PULSE\_WIDTH\_03\_05\_MS
  - Start-up control configuration options (0x73), [971](#)
- RLC\_PULSE\_WIDTH\_10\_12\_MS
  - Start-up control configuration options (0x73), [971](#)
- RLC\_PULSE\_WIDTH\_20\_25\_MS
  - Start-up control configuration options (0x73), [971](#)
- RTC Driver, [893](#)
  - DAYS\_IN\_A\_LEAP\_YEAR, [899](#)
  - DAYS\_IN\_A\_YEAR, [899](#)
  - HOURS\_IN\_A\_DAY, [899](#)
  - MINS\_IN\_A\_HOUR, [899](#)
  - RTC\_CLK\_SRC\_LPO\_1KHZ, [900](#)
  - RTC\_CLK\_SRC\_OSC\_32KHZ, [900](#)
  - RTC\_CLKOUT\_DISABLED, [900](#)
  - RTC\_CLKOUT\_SRC\_32KHZ, [900](#)
  - RTC\_CLKOUT\_SRC\_TSIC, [900](#)
  - RTC\_CTRL\_REG\_LOCK, [900](#)
  - RTC\_DRV\_ConfigureAlarm, [901](#)
  - RTC\_DRV\_ConfigureFaultInt, [901](#)
  - RTC\_DRV\_ConfigureRegisterLock, [901](#)
  - RTC\_DRV\_ConfigureSecondsInt, [902](#)
  - RTC\_DRV\_ConfigureTimeCompensation, [902](#)
  - RTC\_DRV\_ConvertSecondsToTimeDate, [902](#)
  - RTC\_DRV\_ConvertTimeDateToSeconds, [902](#)
  - RTC\_DRV\_Deinit, [903](#)
  - RTC\_DRV\_GetAlarmConfig, [903](#)
  - RTC\_DRV\_GetCurrentTimeDate, [903](#)
  - RTC\_DRV\_GetDefaultConfig, [903](#)
  - RTC\_DRV\_GetNextAlarmTime, [904](#)
  - RTC\_DRV\_GetRegisterLock, [904](#)
  - RTC\_DRV\_GetTimeCompensation, [904](#)
  - RTC\_DRV\_IRQHandler, [905](#)
  - RTC\_DRV\_Init, [904](#)
  - RTC\_DRV\_IsAlarmPending, [905](#)
  - RTC\_DRV\_IsTimeDateCorrectFormat, [905](#)
  - RTC\_DRV\_IsYearLeap, [905](#)
  - RTC\_DRV\_SecondsIRQHandler, [906](#)
  - RTC\_DRV\_SetTimeDate, [906](#)
  - RTC\_DRV\_StartCounter, [906](#)
  - RTC\_DRV\_StopCounter, [906](#)
  - RTC\_INT\_128HZ, [901](#)
  - RTC\_INT\_16HZ, [901](#)
  - RTC\_INT\_1HZ, [900](#)
  - RTC\_INT\_2HZ, [900](#)
  - RTC\_INT\_32HZ, [901](#)
  - RTC\_INT\_4HZ, [900](#)
  - RTC\_INT\_64HZ, [901](#)
  - RTC\_INT\_8HZ, [901](#)
  - RTC\_LOCK\_REG\_LOCK, [900](#)
  - RTC\_STATUS\_REG\_LOCK, [900](#)
  - RTC\_TCL\_REG\_LOCK, [900](#)
  - rtc\_clk\_out\_config\_t, [900](#)
  - rtc\_clk\_select\_t, [900](#)
  - rtc\_lock\_register\_select\_t, [900](#)
  - rtc\_second\_int\_cfg\_t, [900](#)
  - SECONDS\_IN\_A\_DAY, [899](#)
  - SECONDS\_IN\_A\_HOUR, [899](#)
  - SECONDS\_IN\_A\_MIN, [899](#)
  - YEAR\_RANGE\_END, [900](#)
  - YEAR\_RANGE\_START, [900](#)
- RTC\_CLK\_SRC\_LPO\_1KHZ
  - RTC Driver, [900](#)
- RTC\_CLK\_SRC\_OSC\_32KHZ
  - RTC Driver, [900](#)
- RTC\_CLKOUT\_DISABLED
  - RTC Driver, [900](#)
- RTC\_CLKOUT\_SRC\_32KHZ
  - RTC Driver, [900](#)
- RTC\_CLKOUT\_SRC\_TSIC
  - RTC Driver, [900](#)
- RTC\_CTRL\_REG\_LOCK
  - RTC Driver, [900](#)
- RTC\_DRV\_ConfigureAlarm
  - RTC Driver, [901](#)
- RTC\_DRV\_ConfigureFaultInt
  - RTC Driver, [901](#)
- RTC\_DRV\_ConfigureRegisterLock
  - RTC Driver, [901](#)
- RTC\_DRV\_ConfigureSecondsInt
  - RTC Driver, [902](#)
- RTC\_DRV\_ConfigureTimeCompensation
  - RTC Driver, [902](#)
- RTC\_DRV\_ConvertSecondsToTimeDate
  - RTC Driver, [902](#)
- RTC\_DRV\_ConvertTimeDateToSeconds
  - RTC Driver, [902](#)
- RTC\_DRV\_Deinit
  - RTC Driver, [903](#)
- RTC\_DRV\_GetAlarmConfig
  - RTC Driver, [903](#)
- RTC\_DRV\_GetCurrentTimeDate
  - RTC Driver, [903](#)

RTC\_DRV\_GetDefaultConfig  
     RTC Driver, [903](#)  
 RTC\_DRV\_GetNextAlarmTime  
     RTC Driver, [904](#)  
 RTC\_DRV\_GetRegisterLock  
     RTC Driver, [904](#)  
 RTC\_DRV\_GetTimeCompensation  
     RTC Driver, [904](#)  
 RTC\_DRV\_IRQHandler  
     RTC Driver, [905](#)  
 RTC\_DRV\_Init  
     RTC Driver, [904](#)  
 RTC\_DRV\_IsAlarmPending  
     RTC Driver, [905](#)  
 RTC\_DRV\_IsTimeDateCorrectFormat  
     RTC Driver, [905](#)  
 RTC\_DRV\_IsYearLeap  
     RTC Driver, [905](#)  
 RTC\_DRV\_SecondsIRQHandler  
     RTC Driver, [906](#)  
 RTC\_DRV\_SetTimeDate  
     RTC Driver, [906](#)  
 RTC\_DRV\_StartCounter  
     RTC Driver, [906](#)  
 RTC\_DRV\_StopCounter  
     RTC Driver, [906](#)  
 RTC\_INT\_128HZ  
     RTC Driver, [901](#)  
 RTC\_INT\_16HZ  
     RTC Driver, [901](#)  
 RTC\_INT\_1HZ  
     RTC Driver, [900](#)  
 RTC\_INT\_2HZ  
     RTC Driver, [900](#)  
 RTC\_INT\_32HZ  
     RTC Driver, [901](#)  
 RTC\_INT\_4HZ  
     RTC Driver, [900](#)  
 RTC\_INT\_64HZ  
     RTC Driver, [901](#)  
 RTC\_INT\_8HZ  
     RTC Driver, [901](#)  
 RTC\_LOCK\_REG\_LOCK  
     RTC Driver, [900](#)  
 RTC\_STATUS\_REG\_LOCK  
     RTC Driver, [900](#)  
 RTC\_TCL\_REG\_LOCK  
     RTC Driver, [900](#)  
 RUN\_MODE  
     Clock Manager Driver, [305](#)  
 range  
     scg\_firc\_config\_t, [292](#)  
     scg\_sirc\_config\_t, [291](#)  
     scg\_sosc\_config\_t, [290](#)  
 Raw API, [908](#)  
     ld\_get\_raw, [908](#)  
     ld\_put\_raw, [908](#)  
     ld\_raw\_rx\_status, [908](#)  
     ld\_raw\_tx\_status, [909](#)  
 rcRegulatorCfg  
     uja113x\_init\_cfg\_t, [146](#)  
 rccrConfig  
     scg\_clock\_mode\_config\_t, [294](#)  
 rcm\_source\_names\_t  
     Power\_s32k1xx, [879](#)  
 Real Time Clock Driver (RTC), [910](#)  
 receive\_NAD\_ptr  
     lin\_tl\_descriptor\_t, [764](#)  
 receive\_message\_length\_ptr  
     lin\_tl\_descriptor\_t, [764](#)  
 receive\_message\_ptr  
     lin\_tl\_descriptor\_t, [764](#)  
 receiveStatus  
     lpuart\_state\_t, [733](#)  
 refClk  
     clock\_source\_config\_t, [300](#)  
 refFreq  
     clock\_source\_config\_t, [300](#)  
 refIn  
     crc\_spec\_t, [208](#)  
 refOut  
     crc\_spec\_t, [209](#)  
 Register address map, [914](#)  
     B0DRVS\_ADDRESS, [918](#)  
     B0FIE\_ADDRESS, [918](#)  
     B0FIS\_ADDRESS, [918](#)  
     B0OLDTC\_ADDRESS, [918](#)  
     B0SCDTC\_ADDRESS, [918](#)  
     B0WUPIE\_ADDRESS, [918](#)  
     B0WUPIS\_ADDRESS, [918](#)  
     B0WUPS\_ADDRESS, [918](#)  
     B0WUPTHRC\_ADDRESS, [919](#)  
     B1DRVS\_ADDRESS, [919](#)  
     B1FIE\_ADDRESS, [919](#)  
     B1FIS\_ADDRESS, [919](#)  
     B1OLDTC\_ADDRESS, [919](#)  
     B1SCDTC\_ADDRESS, [919](#)  
     B1WUPIE\_ADDRESS, [919](#)  
     B1WUPIS\_ADDRESS, [919](#)  
     B1WUPS\_ADDRESS, [919](#)  
     B1WUPTHRC\_ADDRESS, [919](#)  
     BMETSC\_ADDRESS, [919](#)  
     BMHC\_ADDRESS, [920](#)  
     BMOTC\_ADDRESS, [920](#)  
     BMUTC\_ADDRESS, [920](#)  
     BSADC1\_ADDRESS, [920](#)  
     BSADC2\_ADDRESS, [920](#)  
     CANC\_ADDRESS, [920](#)  
     DID1\_ADDRESS, [920](#)  
     DID2\_ADDRESS, [920](#)  
     DM0\_ADDRESS, [920](#)  
     DM1\_ADDRESS, [920](#)  
     DM2\_ADDRESS, [920](#)  
     DM3\_ADDRESS, [921](#)  
     DM4\_ADDRESS, [921](#)  
     DM5\_ADDRESS, [921](#)

DM6\_ADDRESS, [921](#)  
 DM7\_ADDRESS, [921](#)  
 DR\_ADDRESS, [921](#)  
 FC\_ADDRESS, [921](#)  
 FSC\_ADDRESS, [921](#)  
 GIS\_ADDRESS, [921](#)  
 HVIO1C\_ADDRESS, [921](#)  
 HVIO2C\_ADDRESS, [921](#)  
 HVIO3C\_ADDRESS, [922](#)  
 HVIO4C\_ADDRESS, [922](#)  
 HVIO5C\_ADDRESS, [922](#)  
 HVIO6C\_ADDRESS, [922](#)  
 HVIO7C\_ADDRESS, [922](#)  
 HVIO8C\_ADDRESS, [922](#)  
 HVIOHSDRVC\_ADDRESS, [922](#)  
 HVIOLSDRVC\_ADDRESS, [922](#)  
 ID0\_ADDRESS, [922](#)  
 ID1\_ADDRESS, [922](#)  
 ID2\_ADDRESS, [922](#)  
 ID3\_ADDRESS, [923](#)  
 IDM0\_ADDRESS, [923](#)  
 IDM1\_ADDRESS, [923](#)  
 IDM2\_ADDRESS, [923](#)  
 IDM3\_ADDRESS, [923](#)  
 LCKC\_ADDRESS, [923](#)  
 LINC\_ADDRESS, [923](#)  
 MC\_ADDRESS, [923](#)  
 MEM0\_ADDRESS, [923](#)  
 MEM1\_ADDRESS, [923](#)  
 MEM2\_ADDRESS, [923](#)  
 MEM3\_ADDRESS, [924](#)  
 MSBCS\_ADDRESS, [924](#)  
 MTPNVCRCC\_ADDRESS, [924](#)  
 MTPNVS\_ADDRESS, [924](#)  
 RC\_ADDRESS, [924](#)  
 SBCCC\_ADDRESS, [924](#)  
 SMPSC\_ADDRESS, [924](#)  
 SMPSOVC\_ADDRESS, [924](#)  
 SUPC\_ADDRESS, [924](#)  
 SUPIE\_ADDRESS, [924](#)  
 SUPIS\_ADDRESS, [924](#)  
 SUPVS\_ADDRESS, [925](#)  
 SYSIE\_ADDRESS, [925](#)  
 SYSIS\_ADDRESS, [925](#)  
 TIE\_ADDRESS, [925](#)  
 TIM1C\_ADDRESS, [925](#)  
 TIM1DCC\_ADDRESS, [925](#)  
 TIM2C\_ADDRESS, [925](#)  
 TIM2DCC\_ADDRESS, [925](#)  
 TIM3C\_ADDRESS, [925](#)  
 TIM3DCC\_ADDRESS, [925](#)  
 TIM4C\_ADDRESS, [925](#)  
 TIM4DCC\_ADDRESS, [926](#)  
 TIS\_ADDRESS, [926](#)  
 TS\_ADDRESS, [926](#)  
 VBADC1\_ADDRESS, [926](#)  
 VBADC2\_ADDRESS, [926](#)  
 WTDC\_ADDRESS, [926](#)  
 WTDS\_ADDRESS, [926](#)  
 regulator  
     sbc\_regulator\_ctr\_t, [1019](#)  
     scg\_firc\_config\_t, [292](#)  
 regulatorCtr  
     sbc\_int\_config\_t, [1020](#)  
 repeatForever  
     rtc\_alarm\_config\_t, [897](#)  
 repetitionInterval  
     rtc\_alarm\_config\_t, [897](#)  
 reserved  
     lin\_word\_status\_str\_t, [756](#)  
 resetCntCtrl  
     uja113x\_fsc\_cfg\_t, [179](#)  
 resolution  
     adc\_converter\_config\_t, [217](#)  
     extension\_adc\_s32k1xx\_t, [237](#)  
 resp\_err\_frm\_id\_ptr  
     lin\_node\_attribute\_t, [758](#)  
 response\_buffer\_ptr  
     lin\_protocol\_state\_t, [770](#)  
 response\_error  
     lin\_node\_attribute\_t, [759](#)  
 response\_error\_bit\_offset\_ptr  
     lin\_node\_attribute\_t, [759](#)  
 response\_error\_byte\_offset\_ptr  
     lin\_node\_attribute\_t, [759](#)  
 response\_length  
     lin\_protocol\_state\_t, [770](#)  
 resultBuffer  
     adc\_group\_config\_t, [235](#)  
 resultBufferTail  
     adc\_callback\_info\_t, [1081](#)  
 rlc  
     sbc\_start\_up\_t, [1015](#)  
 roundRobinChannelsState  
     cmp\_trigger\_mode\_t, [331](#)  
 roundRobinInterruptState  
     cmp\_trigger\_mode\_t, [331](#)  
 roundRobinState  
     cmp\_trigger\_mode\_t, [331](#)  
 rss  
     sbc\_main\_status\_t, [1022](#)  
 rtc\_alarm\_config\_t, [896](#)  
     alarmCallback, [897](#)  
     alarmIntEnable, [897](#)  
     alarmTime, [897](#)  
     callbackParams, [897](#)  
     numberOfRepeats, [897](#)  
     repeatForever, [897](#)  
     repetitionInterval, [897](#)  
 rtc\_clk\_out\_config\_t  
     RTC Driver, [900](#)  
 rtc\_clk\_select\_t  
     RTC Driver, [900](#)  
 rtc\_init\_config\_t, [895](#)  
     clockOutConfig, [896](#)  
     clockSelect, [896](#)

- compensation, [896](#)
- compensationInterval, [896](#)
- nonSupervisorAccessEnable, [896](#)
- updateEnable, [896](#)
- rtc\_interrupt\_config\_t, [897](#)
  - callbackParams, [897](#)
  - overflowIntEnable, [898](#)
  - rtcCallback, [898](#)
  - timeInvalidIntEnable, [898](#)
- rtc\_lock\_register\_select\_t
  - RTC Driver, [900](#)
- rtc\_register\_lock\_config\_t, [898](#)
  - controlRegisterLock, [899](#)
  - lockRegisterLock, [899](#)
  - statusRegisterLock, [899](#)
  - timeCompensationRegisterLock, [899](#)
- rtc\_second\_int\_cfg\_t
  - RTC Driver, [900](#)
- rtc\_seconds\_int\_config\_t, [898](#)
  - rtcSecondsCallback, [898](#)
  - secondIntConfig, [898](#)
  - secondIntEnable, [898](#)
  - secondsCallbackParams, [898](#)
- rtc\_timedate\_t, [895](#)
  - day, [895](#)
  - hour, [895](#)
  - minutes, [895](#)
  - month, [895](#)
  - seconds, [895](#)
  - year, [895](#)
- rtcCallback
  - rtc\_interrupt\_config\_t, [898](#)
- rtcClkInFreq
  - scg\_rtc\_config\_t, [293](#)
- rtcConfig
  - scg\_config\_t, [295](#)
- rtcSecondsCallback
  - rtc\_seconds\_int\_config\_t, [898](#)
- rx\_msg\_size
  - lin\_tl\_descriptor\_t, [764](#)
- rx\_msg\_status
  - lin\_tl\_descriptor\_t, [765](#)
- rxBuff
  - lin\_state\_t, [656](#)
  - lpspi\_state\_t, [706](#)
  - lpuart\_state\_t, [733](#)
- rxCallback
  - lpuart\_state\_t, [733](#)
  - uart\_user\_config\_t, [1052](#)
- rxCallbackParam
  - lpuart\_state\_t, [734](#)
  - uart\_user\_config\_t, [1052](#)
- rxComplete
  - lpuart\_state\_t, [734](#)
- rxCompleted
  - lin\_state\_t, [656](#)
- rxCount
  - lpspi\_state\_t, [706](#)
- rxDMAChannel
  - flexio\_i2c\_master\_user\_config\_t, [468](#)
  - flexio\_i2s\_master\_user\_config\_t, [478](#)
  - flexio\_i2s\_slave\_user\_config\_t, [479](#)
  - flexio\_spi\_master\_user\_config\_t, [496](#)
  - flexio\_spi\_slave\_user\_config\_t, [498](#)
  - i2s\_user\_config\_t, [605](#)
  - lpspi\_master\_config\_t, [704](#)
  - lpspi\_slave\_config\_t, [708](#)
  - lpspi\_state\_t, [707](#)
  - lpuart\_user\_config\_t, [735](#)
  - spi\_master\_t, [958](#)
  - spi\_slave\_t, [959](#)
  - uart\_user\_config\_t, [1052](#)
- rxFrameCnt
  - lpspi\_state\_t, [707](#)
- rxPin
  - extension\_flexio\_for\_i2s\_t, [605](#)
  - flexio\_i2s\_master\_user\_config\_t, [478](#)
  - flexio\_i2s\_slave\_user\_config\_t, [479](#)
- rxSize
  - lin\_state\_t, [656](#)
  - lpuart\_state\_t, [734](#)
- S32K142 SoC Header file, [927](#)
- S32K142 System Files, [928](#)
- SADDR
  - edma\_software\_tcd\_t, [383](#)
- SAVE\_CONFIG\_SET
  - Common Core API., [317](#)
- SBC configuration control options (0x74), [929](#)
  - FNMC\_DISABLE, [930](#)
  - FNMC\_ENABLE, [930](#)
  - SDMC\_DISABLE, [930](#)
  - SDMC\_ENABLE, [930](#)
  - SLPC\_SLEEP\_NOT\_SUPPORTED, [930](#)
  - SLPC\_SLEEP\_SUPPORTED, [930](#)
  - V1RTSUC\_60\_NOM, [930](#)
  - V1RTSUC\_70\_NOM, [930](#)
  - V1RTSUC\_80\_NOM, [930](#)
  - V1RTSUC\_90\_NOM, [930](#)
  - VEXTAC\_NO\_PROTECTION, [931](#)
  - VEXTAC\_SENSOR\_SUPPLY, [931](#)
- SBC\_UJA\_CAN
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_CAN\_CFDC\_DIS
  - UJA116xA SBC Driver, [1030](#)
- SBC\_UJA\_CAN\_CFDC\_EN
  - UJA116xA SBC Driver, [1030](#)
- SBC\_UJA\_CAN\_CMC\_ACMODE\_DA
  - UJA116xA SBC Driver, [1030](#)
- SBC\_UJA\_CAN\_CMC\_ACMODE\_DD
  - UJA116xA SBC Driver, [1030](#)
- SBC\_UJA\_CAN\_CMC\_LISTEN
  - UJA116xA SBC Driver, [1030](#)
- SBC\_UJA\_CAN\_CMC\_OFMODE
  - UJA116xA SBC Driver, [1030](#)
- SBC\_UJA\_CAN\_CPNC\_DIS
  - UJA116xA SBC Driver, [1031](#)

- SBC\_UJA\_CAN\_CPNC\_EN
  - UJA116xA SBC Driver, [1031](#)
- SBC\_UJA\_CAN\_PNCOK\_DIS
  - UJA116xA SBC Driver, [1031](#)
- SBC\_UJA\_CAN\_PNCOK\_EN
  - UJA116xA SBC Driver, [1031](#)
- SBC\_UJA\_COUNT\_DMASK
  - UJA116xA SBC Driver, [1029](#)
- SBC\_UJA\_COUNT\_ID\_REG
  - UJA116xA SBC Driver, [1029](#)
- SBC\_UJA\_COUNT\_MASK
  - UJA116xA SBC Driver, [1029](#)
- SBC\_UJA\_DAT\_MASK\_0
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_DAT\_MASK\_1
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_DAT\_MASK\_2
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_DAT\_MASK\_3
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_DAT\_MASK\_4
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_DAT\_MASK\_5
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_DAT\_MASK\_6
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_DAT\_MASK\_7
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_DAT\_RATE
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_DAT\_RATE\_CDR\_1000KB
  - UJA116xA SBC Driver, [1031](#)
- SBC\_UJA\_DAT\_RATE\_CDR\_100KB
  - UJA116xA SBC Driver, [1031](#)
- SBC\_UJA\_DAT\_RATE\_CDR\_125KB
  - UJA116xA SBC Driver, [1031](#)
- SBC\_UJA\_DAT\_RATE\_CDR\_250KB
  - UJA116xA SBC Driver, [1031](#)
- SBC\_UJA\_DAT\_RATE\_CDR\_500KB
  - UJA116xA SBC Driver, [1031](#)
- SBC\_UJA\_DAT\_RATE\_CDR\_50KB
  - UJA116xA SBC Driver, [1031](#)
- SBC\_UJA\_FAIL\_SAFE
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_FAIL\_SAFE\_LHC\_FLOAT
  - UJA116xA SBC Driver, [1031](#)
- SBC\_UJA\_FAIL\_SAFE\_LHC\_LOW
  - UJA116xA SBC Driver, [1031](#)
- SBC\_UJA\_FRAME\_CTR
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_FRAME\_CTR\_IDE\_11B
  - UJA116xA SBC Driver, [1032](#)
- SBC\_UJA\_FRAME\_CTR\_IDE\_29B
  - UJA116xA SBC Driver, [1032](#)
- SBC\_UJA\_FRAME\_CTR\_PNDM\_DCARE
  - UJA116xA SBC Driver, [1032](#)
- SBC\_UJA\_FRAME\_CTR\_PNDM\_EVAL
  - UJA116xA SBC Driver, [1032](#)
- SBC\_UJA\_GL\_EVNT\_STAT
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_GL\_EVNT\_STAT\_SUPE
  - UJA116xA SBC Driver, [1032](#)
- SBC\_UJA\_GL\_EVNT\_STAT\_SUPE\_NO
  - UJA116xA SBC Driver, [1032](#)
- SBC\_UJA\_GL\_EVNT\_STAT\_SYSE
  - UJA116xA SBC Driver, [1032](#)
- SBC\_UJA\_GL\_EVNT\_STAT\_SYSE\_NO
  - UJA116xA SBC Driver, [1032](#)
- SBC\_UJA\_GL\_EVNT\_STAT\_TRXE
  - UJA116xA SBC Driver, [1032](#)
- SBC\_UJA\_GL\_EVNT\_STAT\_TRXE\_NO
  - UJA116xA SBC Driver, [1032](#)
- SBC\_UJA\_GL\_EVNT\_STAT\_WPE
  - UJA116xA SBC Driver, [1033](#)
- SBC\_UJA\_GL\_EVNT\_STAT\_WPE\_NO
  - UJA116xA SBC Driver, [1033](#)
- SBC\_UJA\_IDENTIF
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_IDENTIF\_0
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_IDENTIF\_1
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_IDENTIF\_2
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_IDENTIF\_3
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_LOCK
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_MAIN
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_MAIN\_NMS\_NORMAL
  - UJA116xA SBC Driver, [1033](#)
- SBC\_UJA\_MAIN\_NMS\_PWR\_UP
  - UJA116xA SBC Driver, [1033](#)
- SBC\_UJA\_MAIN\_OTWS\_ABOVE
  - UJA116xA SBC Driver, [1033](#)
- SBC\_UJA\_MAIN\_OTWS\_BELOW
  - UJA116xA SBC Driver, [1033](#)
- SBC\_UJA\_MAIN\_RSS\_CAN\_WAKEUP
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MAIN\_RSS\_DIAG\_WAKEUP
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MAIN\_RSS\_ILLEG\_SLP
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MAIN\_RSS\_ILLEG\_WATCH
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MAIN\_RSS\_LFT\_OVERTM
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MAIN\_RSS\_OFF\_MODE
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MAIN\_RSS\_OVF\_SLP
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MAIN\_RSS\_RSTN\_PULDW
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MAIN\_RSS\_SLP\_WAKEUP
  - UJA116xA SBC Driver, [1034](#)



- SBC\_UJA\_MAIN\_RSS\_V1\_UNDERV
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MAIN\_RSS\_WAKE\_SLP
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MAIN\_RSS\_WATCH\_OVF
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MAIN\_RSS\_WATCH\_TRIG
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MASK\_0
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_MASK\_1
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_MASK\_2
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_MASK\_3
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_MEMORY\_0
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_MEMORY\_1
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_MEMORY\_2
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_MEMORY\_3
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_MODE
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_MODE\_MC\_NORMAL
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MODE\_MC\_SLEEP
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MODE\_MC\_STANDBY
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MTPNV\_CRC
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_MTPNV\_STAT
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_MTPNV\_STAT\_ECCS
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MTPNV\_STAT\_ECCS\_NO
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MTPNV\_STAT\_NVMPs
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_MTPNV\_STAT\_NVMPs\_NO
  - UJA116xA SBC Driver, [1034](#)
- SBC\_UJA\_REGULATOR
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_REGULATOR\_PDC\_HV
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_REGULATOR\_PDC\_LV
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_REGULATOR\_V1RTC\_60
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_REGULATOR\_V1RTC\_70
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_REGULATOR\_V1RTC\_80
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_REGULATOR\_V1RTC\_90
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_REGULATOR\_V2C\_N
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_REGULATOR\_V2C\_N\_S\_R
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_REGULATOR\_V2C\_N\_S\_S\_R
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_REGULATOR\_V2C\_OFF
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_SBC
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_SBC\_FNMC\_DIS
  - UJA116xA SBC Driver, [1037](#)
- SBC\_UJA\_SBC\_FNMC\_EN
  - UJA116xA SBC Driver, [1037](#)
- SBC\_UJA\_SBC\_SDMC\_DIS
  - UJA116xA SBC Driver, [1037](#)
- SBC\_UJA\_SBC\_SDMC\_EN
  - UJA116xA SBC Driver, [1037](#)
- SBC\_UJA\_SBC\_SLPC\_AC
  - UJA116xA SBC Driver, [1037](#)
- SBC\_UJA\_SBC\_SLPC\_IG
  - UJA116xA SBC Driver, [1037](#)
- SBC\_UJA\_SBC\_V1RTSUC\_60
  - UJA116xA SBC Driver, [1037](#)
- SBC\_UJA\_SBC\_V1RTSUC\_70
  - UJA116xA SBC Driver, [1037](#)
- SBC\_UJA\_SBC\_V1RTSUC\_80
  - UJA116xA SBC Driver, [1037](#)
- SBC\_UJA\_SBC\_V1RTSUC\_90
  - UJA116xA SBC Driver, [1037](#)
- SBC\_UJA\_START\_UP
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_START\_UP\_RLC\_01\_01p5
  - UJA116xA SBC Driver, [1038](#)
- SBC\_UJA\_START\_UP\_RLC\_03p6\_05
  - UJA116xA SBC Driver, [1038](#)
- SBC\_UJA\_START\_UP\_RLC\_10\_12p5
  - UJA116xA SBC Driver, [1038](#)
- SBC\_UJA\_START\_UP\_RLC\_20\_25p0
  - UJA116xA SBC Driver, [1038](#)
- SBC\_UJA\_START\_UP\_V2SUC\_00
  - UJA116xA SBC Driver, [1038](#)
- SBC\_UJA\_START\_UP\_V2SUC\_11
  - UJA116xA SBC Driver, [1038](#)
- SBC\_UJA\_SUP\_EVNT\_STAT
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_SUP\_EVNT\_STAT\_V1U
  - UJA116xA SBC Driver, [1038](#)
- SBC\_UJA\_SUP\_EVNT\_STAT\_V1U\_NO
  - UJA116xA SBC Driver, [1038](#)
- SBC\_UJA\_SUP\_EVNT\_STAT\_V2O
  - UJA116xA SBC Driver, [1038](#)
- SBC\_UJA\_SUP\_EVNT\_STAT\_V2O\_NO
  - UJA116xA SBC Driver, [1038](#)
- SBC\_UJA\_SUP\_EVNT\_STAT\_V2U
  - UJA116xA SBC Driver, [1038](#)
- SBC\_UJA\_SUP\_EVNT\_STAT\_V2U\_NO
  - UJA116xA SBC Driver, [1038](#)

SBC\_UJA\_SUPPLY\_EVNT  
     UJA116xA SBC Driver, [1035](#)  
 SBC\_UJA\_SUPPLY\_EVNT\_V1UE\_DIS  
     UJA116xA SBC Driver, [1039](#)  
 SBC\_UJA\_SUPPLY\_EVNT\_V1UE\_EN  
     UJA116xA SBC Driver, [1039](#)  
 SBC\_UJA\_SUPPLY\_EVNT\_V2OE\_DIS  
     UJA116xA SBC Driver, [1039](#)  
 SBC\_UJA\_SUPPLY\_EVNT\_V2OE\_EN  
     UJA116xA SBC Driver, [1039](#)  
 SBC\_UJA\_SUPPLY\_EVNT\_V2UE\_DIS  
     UJA116xA SBC Driver, [1039](#)  
 SBC\_UJA\_SUPPLY\_EVNT\_V2UE\_EN  
     UJA116xA SBC Driver, [1039](#)  
 SBC\_UJA\_SUPPLY\_STAT  
     UJA116xA SBC Driver, [1035](#)  
 SBC\_UJA\_SUPPLY\_STAT\_V1S\_VAB  
     UJA116xA SBC Driver, [1039](#)  
 SBC\_UJA\_SUPPLY\_STAT\_V1S\_VBE  
     UJA116xA SBC Driver, [1039](#)  
 SBC\_UJA\_SUPPLY\_STAT\_V2S\_DIS  
     UJA116xA SBC Driver, [1039](#)  
 SBC\_UJA\_SUPPLY\_STAT\_V2S\_VAB  
     UJA116xA SBC Driver, [1039](#)  
 SBC\_UJA\_SUPPLY\_STAT\_V2S\_VBE  
     UJA116xA SBC Driver, [1039](#)  
 SBC\_UJA\_SUPPLY\_STAT\_V2S\_VOK  
     UJA116xA SBC Driver, [1039](#)  
 SBC\_UJA\_SYS\_EVNT\_OTWE\_DIS  
     UJA116xA SBC Driver, [1040](#)  
 SBC\_UJA\_SYS\_EVNT\_OTWE\_EN  
     UJA116xA SBC Driver, [1040](#)  
 SBC\_UJA\_SYS\_EVNT\_SPIFE\_DIS  
     UJA116xA SBC Driver, [1040](#)  
 SBC\_UJA\_SYS\_EVNT\_SPIFE\_EN  
     UJA116xA SBC Driver, [1040](#)  
 SBC\_UJA\_SYS\_EVNT\_STAT  
     UJA116xA SBC Driver, [1035](#)  
 SBC\_UJA\_SYS\_EVNT\_STAT\_OTW  
     UJA116xA SBC Driver, [1040](#)  
 SBC\_UJA\_SYS\_EVNT\_STAT\_OTW\_NO  
     UJA116xA SBC Driver, [1040](#)  
 SBC\_UJA\_SYS\_EVNT\_STAT\_PO  
     UJA116xA SBC Driver, [1040](#)  
 SBC\_UJA\_SYS\_EVNT\_STAT\_PO\_NO  
     UJA116xA SBC Driver, [1040](#)  
 SBC\_UJA\_SYS\_EVNT\_STAT\_SPIF  
     UJA116xA SBC Driver, [1040](#)  
 SBC\_UJA\_SYS\_EVNT\_STAT\_SPIF\_NO  
     UJA116xA SBC Driver, [1040](#)  
 SBC\_UJA\_SYS\_EVNT\_STAT\_WDF  
     UJA116xA SBC Driver, [1041](#)  
 SBC\_UJA\_SYS\_EVNT\_STAT\_WDF\_NO  
     UJA116xA SBC Driver, [1041](#)  
 SBC\_UJA\_SYSTEM\_EVNT  
     UJA116xA SBC Driver, [1035](#)  
 SBC\_UJA\_TIMEOUT  
     UJA116xA SBC Driver, [1029](#)

SBC\_UJA\_TRANS\_EVNT  
     UJA116xA SBC Driver, [1035](#)  
 SBC\_UJA\_TRANS\_EVNT\_CBSE\_DIS  
     UJA116xA SBC Driver, [1041](#)  
 SBC\_UJA\_TRANS\_EVNT\_CBSE\_EN  
     UJA116xA SBC Driver, [1041](#)  
 SBC\_UJA\_TRANS\_EVNT\_CFE\_DIS  
     UJA116xA SBC Driver, [1041](#)  
 SBC\_UJA\_TRANS\_EVNT\_CFE\_EN  
     UJA116xA SBC Driver, [1041](#)  
 SBC\_UJA\_TRANS\_EVNT\_CWE\_DIS  
     UJA116xA SBC Driver, [1041](#)  
 SBC\_UJA\_TRANS\_EVNT\_CWE\_EN  
     UJA116xA SBC Driver, [1041](#)  
 SBC\_UJA\_TRANS\_EVNT\_STAT  
     UJA116xA SBC Driver, [1035](#)  
 SBC\_UJA\_TRANS\_EVNT\_STAT\_CBS  
     UJA116xA SBC Driver, [1041](#)  
 SBC\_UJA\_TRANS\_EVNT\_STAT\_CBS\_NO  
     UJA116xA SBC Driver, [1041](#)  
 SBC\_UJA\_TRANS\_EVNT\_STAT\_CF  
     UJA116xA SBC Driver, [1042](#)  
 SBC\_UJA\_TRANS\_EVNT\_STAT\_CF\_NO  
     UJA116xA SBC Driver, [1042](#)  
 SBC\_UJA\_TRANS\_EVNT\_STAT\_CW  
     UJA116xA SBC Driver, [1042](#)  
 SBC\_UJA\_TRANS\_EVNT\_STAT\_CW\_NO  
     UJA116xA SBC Driver, [1042](#)  
 SBC\_UJA\_TRANS\_EVNT\_STAT\_PNFDE  
     UJA116xA SBC Driver, [1042](#)  
 SBC\_UJA\_TRANS\_EVNT\_STAT\_PNFDE\_NO  
     UJA116xA SBC Driver, [1042](#)  
 SBC\_UJA\_TRANS\_STAT  
     UJA116xA SBC Driver, [1035](#)  
 SBC\_UJA\_TRANS\_STAT\_CBSS\_ACT  
     UJA116xA SBC Driver, [1042](#)  
 SBC\_UJA\_TRANS\_STAT\_CBSS\_INACT  
     UJA116xA SBC Driver, [1042](#)  
 SBC\_UJA\_TRANS\_STAT\_CFS\_NO\_TXD  
     UJA116xA SBC Driver, [1043](#)  
 SBC\_UJA\_TRANS\_STAT\_CFS\_TXD  
     UJA116xA SBC Driver, [1043](#)  
 SBC\_UJA\_TRANS\_STAT\_COSCS\_NRUN  
     UJA116xA SBC Driver, [1043](#)  
 SBC\_UJA\_TRANS\_STAT\_COSCS\_RUN  
     UJA116xA SBC Driver, [1043](#)  
 SBC\_UJA\_TRANS\_STAT\_CPNERR\_DET  
     UJA116xA SBC Driver, [1043](#)  
 SBC\_UJA\_TRANS\_STAT\_CPNERR\_NO\_DET  
     UJA116xA SBC Driver, [1043](#)  
 SBC\_UJA\_TRANS\_STAT\_CPNS\_ERR  
     UJA116xA SBC Driver, [1043](#)  
 SBC\_UJA\_TRANS\_STAT\_CPNS\_OK  
     UJA116xA SBC Driver, [1043](#)  
 SBC\_UJA\_TRANS\_STAT\_CTS\_ACT  
     UJA116xA SBC Driver, [1043](#)  
 SBC\_UJA\_TRANS\_STAT\_CTS\_INACT  
     UJA116xA SBC Driver, [1043](#)



- SBC\_UJA\_TRANS\_STAT\_VCS\_AB
  - UJA116xA SBC Driver, [1044](#)
- SBC\_UJA\_TRANS\_STAT\_VCS\_BE
  - UJA116xA SBC Driver, [1044](#)
- SBC\_UJA\_WAKE\_EN
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_WAKE\_EN\_WPFE\_DIS
  - UJA116xA SBC Driver, [1044](#)
- SBC\_UJA\_WAKE\_EN\_WPFE\_EN
  - UJA116xA SBC Driver, [1044](#)
- SBC\_UJA\_WAKE\_EN\_WPRE\_DIS
  - UJA116xA SBC Driver, [1044](#)
- SBC\_UJA\_WAKE\_EN\_WPRE\_EN
  - UJA116xA SBC Driver, [1044](#)
- SBC\_UJA\_WAKE\_EVNT\_STAT
  - UJA116xA SBC Driver, [1036](#)
- SBC\_UJA\_WAKE\_EVNT\_STAT\_WPF
  - UJA116xA SBC Driver, [1044](#)
- SBC\_UJA\_WAKE\_EVNT\_STAT\_WPF\_NO
  - UJA116xA SBC Driver, [1044](#)
- SBC\_UJA\_WAKE\_EVNT\_STAT\_WPR
  - UJA116xA SBC Driver, [1044](#)
- SBC\_UJA\_WAKE\_EVNT\_STAT\_WPR\_NO
  - UJA116xA SBC Driver, [1044](#)
- SBC\_UJA\_WAKE\_STAT
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_WAKE\_STAT\_WPVS\_AB
  - UJA116xA SBC Driver, [1045](#)
- SBC\_UJA\_WAKE\_STAT\_WPVS\_BE
  - UJA116xA SBC Driver, [1045](#)
- SBC\_UJA\_WTDOG\_CTL
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_WTDOG\_CTL\_NWP\_1024
  - UJA116xA SBC Driver, [1045](#)
- SBC\_UJA\_WTDOG\_CTL\_NWP\_128
  - UJA116xA SBC Driver, [1045](#)
- SBC\_UJA\_WTDOG\_CTL\_NWP\_16
  - UJA116xA SBC Driver, [1045](#)
- SBC\_UJA\_WTDOG\_CTL\_NWP\_256
  - UJA116xA SBC Driver, [1045](#)
- SBC\_UJA\_WTDOG\_CTL\_NWP\_32
  - UJA116xA SBC Driver, [1045](#)
- SBC\_UJA\_WTDOG\_CTL\_NWP\_4096
  - UJA116xA SBC Driver, [1045](#)
- SBC\_UJA\_WTDOG\_CTL\_NWP\_64
  - UJA116xA SBC Driver, [1045](#)
- SBC\_UJA\_WTDOG\_CTL\_NWP\_8
  - UJA116xA SBC Driver, [1045](#)
- SBC\_UJA\_WTDOG\_CTL\_WMC\_AUTO
  - UJA116xA SBC Driver, [1045](#)
- SBC\_UJA\_WTDOG\_CTL\_WMC\_TIME
  - UJA116xA SBC Driver, [1045](#)
- SBC\_UJA\_WTDOG\_CTL\_WMC\_WIND
  - UJA116xA SBC Driver, [1045](#)
- SBC\_UJA\_WTDOG\_STAT
  - UJA116xA SBC Driver, [1035](#)
- SBC\_UJA\_WTDOG\_STAT\_FNMS\_N\_NORMAL
  - UJA116xA SBC Driver, [1046](#)
- SBC\_UJA\_WTDOG\_STAT\_FNMS\_NORMAL
  - UJA116xA SBC Driver, [1046](#)
- SBC\_UJA\_WTDOG\_STAT\_SDMS\_N\_NORMAL
  - UJA116xA SBC Driver, [1046](#)
- SBC\_UJA\_WTDOG\_STAT\_SDMS\_NORMAL
  - UJA116xA SBC Driver, [1046](#)
- SBC\_UJA\_WTDOG\_STAT\_WDS\_FIH
  - UJA116xA SBC Driver, [1046](#)
- SBC\_UJA\_WTDOG\_STAT\_WDS\_OFF
  - UJA116xA SBC Driver, [1046](#)
- SBC\_UJA\_WTDOG\_STAT\_WDS\_SEH
  - UJA116xA SBC Driver, [1046](#)
- SBCCC\_ADDRESS
  - Register address map, [924](#)
- SCG\_ASYNC\_CLOCK\_DISABLE
  - Clock Manager Driver, [305](#)
- SCG\_ASYNC\_CLOCK\_DIV\_BY\_1
  - Clock Manager Driver, [305](#)
- SCG\_ASYNC\_CLOCK\_DIV\_BY\_16
  - Clock Manager Driver, [305](#)
- SCG\_ASYNC\_CLOCK\_DIV\_BY\_2
  - Clock Manager Driver, [305](#)
- SCG\_ASYNC\_CLOCK\_DIV\_BY\_32
  - Clock Manager Driver, [305](#)
- SCG\_ASYNC\_CLOCK\_DIV\_BY\_4
  - Clock Manager Driver, [305](#)
- SCG\_ASYNC\_CLOCK\_DIV\_BY\_64
  - Clock Manager Driver, [305](#)
- SCG\_ASYNC\_CLOCK\_DIV\_BY\_8
  - Clock Manager Driver, [305](#)
- SCG\_CLOCKOUT\_SRC\_FIRC
  - Clock Manager Driver, [305](#)
- SCG\_CLOCKOUT\_SRC\_SCG\_SLOW
  - Clock Manager Driver, [305](#)
- SCG\_CLOCKOUT\_SRC\_SIRC
  - Clock Manager Driver, [305](#)
- SCG\_CLOCKOUT\_SRC\_SOSC
  - Clock Manager Driver, [305](#)
- SCG\_CLOCKOUT\_SRC\_SPLL
  - Clock Manager Driver, [305](#)
- SCG\_FIRC\_RANGE\_48M
  - Clock Manager Driver, [306](#)
- SCG\_SIRC\_RANGE\_HIGH
  - Clock Manager Driver, [306](#)
- SCG\_SOSC\_GAIN\_HIGH
  - Clock Manager Driver, [306](#)
- SCG\_SOSC\_GAIN\_LOW
  - Clock Manager Driver, [306](#)
- SCG\_SOSC\_MONITOR\_DISABLE
  - Clock Manager Driver, [306](#)
- SCG\_SOSC\_MONITOR\_INT
  - Clock Manager Driver, [306](#)
- SCG\_SOSC\_MONITOR\_RESET
  - Clock Manager Driver, [306](#)
- SCG\_SOSC\_RANGE\_HIGH
  - Clock Manager Driver, [306](#)
- SCG\_SOSC\_RANGE\_MID
  - Clock Manager Driver, [306](#)

- SCG\_SOSC\_REF\_EXT
  - Clock Manager Driver, [306](#)
- SCG\_SOSC\_REF\_OSC
  - Clock Manager Driver, [306](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_16
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_17
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_18
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_19
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_20
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_21
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_22
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_23
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_24
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_25
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_26
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_27
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_28
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_29
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_30
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_31
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_32
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_33
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_34
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_35
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_36
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_37
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_38
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_39
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_40
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_41
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_42
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_43
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_44
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_45
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_46
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_MULTIPLY\_BY\_47
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_1
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_2
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_3
  - Clock Manager Driver, [307](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_4
  - Clock Manager Driver, [308](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_5
  - Clock Manager Driver, [308](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_6
  - Clock Manager Driver, [308](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_7
  - Clock Manager Driver, [308](#)
- SCG\_SPLL\_CLOCK\_PREDIV\_BY\_8
  - Clock Manager Driver, [308](#)
- SCG\_SPLL\_MONITOR\_DISABLE
  - Clock Manager Driver, [308](#)
- SCG\_SPLL\_MONITOR\_INT
  - Clock Manager Driver, [308](#)
- SCG\_SPLL\_MONITOR\_RESET
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_1
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_10
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_11
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_12
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_13
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_14
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_15
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_16
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_2
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_3
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_4
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_5
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_6
  - Clock Manager Driver, [308](#)

- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_7
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_8
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_DIV\_BY\_9
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_SRC\_FIRC
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_SRC\_NONE
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_SRC\_SIRC
  - Clock Manager Driver, [308](#)
- SCG\_SYSTEM\_CLOCK\_SRC\_SYS\_OSC
  - Clock Manager Driver, [308](#)
- SDMC\_DISABLE
  - SBC configuration control options (0x74), [930](#)
- SDMC\_ENABLE
  - SBC configuration control options (0x74), [930](#)
- SECONDS\_IN\_A\_DAY
  - RTC Driver, [899](#)
- SECONDS\_IN\_A\_HOUR
  - RTC Driver, [899](#)
- SECONDS\_IN\_A\_MIN
  - RTC Driver, [899](#)
- SECURITY\_BOOT\_MAC
  - Security PAL, [937](#)
- SECURITY\_BOOT\_MAC\_KEY
  - Security PAL, [937](#)
- SECURITY\_BOOT\_NOT\_DEFINED
  - Security PAL, [936](#)
- SECURITY\_BOOT\_PARALLEL
  - Security PAL, [935](#)
- SECURITY\_BOOT\_SERIAL
  - Security PAL, [935](#)
- SECURITY\_BOOT\_STRICT
  - Security PAL, [935](#)
- SECURITY\_BootDefine
  - Security PAL, [937](#)
- SECURITY\_BootFailure
  - Security PAL, [937](#)
- SECURITY\_BootOk
  - Security PAL, [938](#)
- SECURITY\_CMD\_BOOT\_DEFINE
  - Security PAL, [936](#)
- SECURITY\_CMD\_BOOT\_FAILURE
  - Security PAL, [936](#)
- SECURITY\_CMD\_BOOT\_OK
  - Security PAL, [936](#)
- SECURITY\_CMD\_DBG\_AUTH
  - Security PAL, [936](#)
- SECURITY\_CMD\_DBG\_CHAL
  - Security PAL, [936](#)
- SECURITY\_CMD\_DEC\_CBC
  - Security PAL, [936](#)
- SECURITY\_CMD\_DEC\_ECB
  - Security PAL, [936](#)
- SECURITY\_CMD\_ENC\_CBC
  - Security PAL, [936](#)
- SECURITY\_CMD\_ENC\_ECB
  - Security PAL, [936](#)
- SECURITY\_CMD\_EXPORT\_RAM\_KEY
  - Security PAL, [936](#)
- SECURITY\_CMD\_EXTEND\_SEED
  - Security PAL, [936](#)
- SECURITY\_CMD\_GENERATE\_MAC
  - Security PAL, [936](#)
- SECURITY\_CMD\_GET\_ID
  - Security PAL, [936](#)
- SECURITY\_CMD\_INIT\_RNG
  - Security PAL, [936](#)
- SECURITY\_CMD\_LOAD\_KEY
  - Security PAL, [936](#)
- SECURITY\_CMD\_LOAD\_PLAIN\_KEY
  - Security PAL, [936](#)
- SECURITY\_CMD\_MP\_COMPRESS
  - Security PAL, [936](#)
- SECURITY\_CMD\_RESERVED\_1
  - Security PAL, [936](#)
- SECURITY\_CMD\_RESERVED\_2
  - Security PAL, [936](#)
- SECURITY\_CMD\_RESERVED\_3
  - Security PAL, [936](#)
- SECURITY\_CMD\_RND
  - Security PAL, [936](#)
- SECURITY\_CMD\_VERIFY\_MAC
  - Security PAL, [936](#)
- SECURITY\_CancelCommand
  - Security PAL, [938](#)
- SECURITY\_DbgAuth
  - Security PAL, [938](#)
- SECURITY\_DbgChal
  - Security PAL, [938](#)
- SECURITY\_DecryptCbc
  - Security PAL, [939](#)
- SECURITY\_DecryptCbcBlocking
  - Security PAL, [939](#)
- SECURITY\_DecryptEcb
  - Security PAL, [940](#)
- SECURITY\_DecryptEcbBlocking
  - Security PAL, [940](#)
- SECURITY\_Deinit
  - Security PAL, [940](#)
- SECURITY\_EncryptCbc
  - Security PAL, [941](#)
- SECURITY\_EncryptCbcBlocking
  - Security PAL, [941](#)
- SECURITY\_EncryptEcb
  - Security PAL, [941](#)
- SECURITY\_EncryptEcbBlocking
  - Security PAL, [942](#)
- SECURITY\_ExportRamKey
  - Security PAL, [942](#)
- SECURITY\_ExtendSeed
  - Security PAL, [943](#)
- SECURITY\_GenerateMac
  - Security PAL, [943](#)

SECURITY\_GenerateMacBlocking  
 Security PAL, [943](#)

SECURITY\_GenerateRnd  
 Security PAL, [945](#)

SECURITY\_GenerateTrnd  
 Security PAL, [945](#)

SECURITY\_GetAsyncCmdStatus  
 Security PAL, [945](#)

SECURITY\_GetDefaultConfig  
 Security PAL, [946](#)

SECURITY\_GetId  
 Security PAL, [946](#)

SECURITY\_INSTANCE0  
 Security PAL, [936](#)

SECURITY\_Init  
 Security PAL, [946](#)

SECURITY\_InitRng  
 Security PAL, [946](#)

SECURITY\_KEY\_1  
 Security PAL, [937](#)

SECURITY\_KEY\_10  
 Security PAL, [937](#)

SECURITY\_KEY\_11  
 Security PAL, [937](#)

SECURITY\_KEY\_12  
 Security PAL, [937](#)

SECURITY\_KEY\_13  
 Security PAL, [937](#)

SECURITY\_KEY\_14  
 Security PAL, [937](#)

SECURITY\_KEY\_15  
 Security PAL, [937](#)

SECURITY\_KEY\_16  
 Security PAL, [937](#)

SECURITY\_KEY\_17  
 Security PAL, [937](#)

SECURITY\_KEY\_2  
 Security PAL, [937](#)

SECURITY\_KEY\_3  
 Security PAL, [937](#)

SECURITY\_KEY\_4  
 Security PAL, [937](#)

SECURITY\_KEY\_5  
 Security PAL, [937](#)

SECURITY\_KEY\_6  
 Security PAL, [937](#)

SECURITY\_KEY\_7  
 Security PAL, [937](#)

SECURITY\_KEY\_8  
 Security PAL, [937](#)

SECURITY\_KEY\_9  
 Security PAL, [937](#)

SECURITY\_LoadKey  
 Security PAL, [948](#)

SECURITY\_LoadPlainKey  
 Security PAL, [948](#)

SECURITY\_MASTER\_ECU  
 Security PAL, [936](#)

SECURITY\_MPCompress  
 Security PAL, [948](#)

SECURITY\_RAM\_KEY  
 Security PAL, [937](#)

SECURITY\_SECRET\_KEY  
 Security PAL, [936](#)

SECURITY\_SecureBoot  
 Security PAL, [949](#)

SECURITY\_VerifyMac  
 Security PAL, [949](#)

SECURITY\_VerifyMacBlocking  
 Security PAL, [950](#)

SERIVCE\_FAULT\_MEMORY\_CLEAR  
 Low level API, [772](#)

SERVICE\_ASSIGN\_FRAME\_ID  
 Low level API, [772](#)

SERVICE\_ASSIGN\_FRAME\_ID\_RANGE  
 Low level API, [772](#)

SERVICE\_ASSIGN\_NAD  
 Low level API, [772](#)

SERVICE\_CONDITIONAL\_CHANGE\_NAD  
 Low level API, [772](#)

SERVICE\_FAULT\_MEMORY\_READ  
 Low level API, [773](#)

SERVICE\_IO\_CONTROL\_BY\_IDENTIFY  
 Low level API, [773](#)

SERVICE\_NOT\_SUPPORTED  
 Common Transport Layer API, [320](#)

SERVICE\_READ\_BY\_IDENTIFY  
 Low level API, [773](#)

SERVICE\_READ\_DATA\_BY\_IDENTIFY  
 Low level API, [773](#)

SERVICE\_SAVE\_CONFIGURATION  
 Low level API, [773](#)

SERVICE\_SESSION\_CONTROL  
 Low level API, [773](#)

SERVICE\_TARGET\_RESET  
 Common Transport Layer API, [320](#)

SERVICE\_WRITE\_DATA\_BY\_IDENTIFY  
 Low level API, [773](#)

SIM\_CLKOUT\_DIV\_BY\_1  
 Clock Manager Driver, [309](#)

SIM\_CLKOUT\_DIV\_BY\_2  
 Clock Manager Driver, [309](#)

SIM\_CLKOUT\_DIV\_BY\_3  
 Clock Manager Driver, [309](#)

SIM\_CLKOUT\_DIV\_BY\_4  
 Clock Manager Driver, [309](#)

SIM\_CLKOUT\_DIV\_BY\_5  
 Clock Manager Driver, [309](#)

SIM\_CLKOUT\_DIV\_BY\_6  
 Clock Manager Driver, [309](#)

SIM\_CLKOUT\_DIV\_BY\_7  
 Clock Manager Driver, [309](#)

SIM\_CLKOUT\_DIV\_BY\_8  
 Clock Manager Driver, [309](#)

SIM\_CLKOUT\_SEL\_SYSTEM\_BUS\_CLK  
 Clock Manager Driver, [309](#)

- SIM\_CLKOUT\_SEL\_SYSTEM\_FIRC\_DIV2\_CLK
  - Clock Manager Driver, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_HCLK
  - Clock Manager Driver, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_LPO\_128K\_CLK
  - Clock Manager Driver, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_LPO\_CLK
  - Clock Manager Driver, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_RTC\_CLK
  - Clock Manager Driver, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_SCG\_CLKOUT
  - Clock Manager Driver, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_SIRC\_DIV2\_CLK
  - Clock Manager Driver, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_SOSC\_DIV2\_CLK
  - Clock Manager Driver, [309](#)
- SIM\_CLKOUT\_SEL\_SYSTEM\_SPLL\_DIV2\_CLK
  - Clock Manager Driver, [309](#)
- SIM\_LPO\_CLK\_SEL\_LPO\_128K
  - Clock Manager Driver, [309](#)
- SIM\_LPO\_CLK\_SEL\_LPO\_1K
  - Clock Manager Driver, [309](#)
- SIM\_LPO\_CLK\_SEL\_LPO\_32K
  - Clock Manager Driver, [309](#)
- SIM\_LPO\_CLK\_SEL\_NO\_CLOCK
  - Clock Manager Driver, [309](#)
- SIM\_RTCCLK\_SEL\_FIRCDIV1\_CLK
  - Clock Manager Driver, [310](#)
- SIM\_RTCCLK\_SEL\_LPO\_32K
  - Clock Manager Driver, [310](#)
- SIM\_RTCCLK\_SEL\_RTC\_CLKIN
  - Clock Manager Driver, [310](#)
- SIM\_RTCCLK\_SEL\_SOSCDIV1\_CLK
  - Clock Manager Driver, [310](#)
- SLAST
  - edma\_software\_tcd\_t, [383](#)
- SLAVE
  - LIN Driver, [657](#)
- SLOW\_CLK\_INDEX
  - Clock Manager Driver, [303](#)
- SLPC\_SLEEP\_NOT\_SUPPORTED
  - SBC configuration control options (0x74), [930](#)
- SLPC\_SLEEP\_SUPPORTED
  - SBC configuration control options (0x74), [930](#)
- SMC\_HSRUN
  - Power\_s32k1xx, [880](#)
- SMC\_RESERVED\_RUN
  - Power\_s32k1xx, [880](#)
- SMC\_RESERVED\_STOP1
  - Power\_s32k1xx, [880](#)
- SMC\_RUN
  - Power\_s32k1xx, [880](#)
- SMC\_STOP
  - Power\_s32k1xx, [880](#)
- SMC\_STOP1
  - Power\_s32k1xx, [880](#)
- SMC\_STOP2
  - Power\_s32k1xx, [880](#)
- SMC\_STOP\_RESERVED
  - Power\_s32k1xx, [880](#)
- SMC\_VLPR
  - Power\_s32k1xx, [880](#)
- SMC\_VLPS
  - Power\_s32k1xx, [880](#)
- SMPSC\_ADDRESS
  - Register address map, [924](#)
- SMPSC\_ON\_NSR
  - 03. SMPS configuration, [183](#)
- SMPSC\_ON\_NSRS
  - 03. SMPS configuration, [183](#)
- SMPSC\_PASS\_THROUGH\_NSS
  - 03. SMPS configuration, [183](#)
- SMPSOC\_50V
  - 03. SMPS configuration, [183](#)
- SMPSOC\_52V
  - 03. SMPS configuration, [183](#)
- SMPSOC\_54V
  - 03. SMPS configuration, [183](#)
- SMPSOC\_56V
  - 03. SMPS configuration, [183](#)
- SMPSOC\_58V
  - 03. SMPS configuration, [184](#)
- SMPSOC\_60V
  - 03. SMPS configuration, [184](#)
- SMPSOC\_62V
  - 03. SMPS configuration, [184](#)
- SMPSOC\_64V
  - 03. SMPS configuration, [184](#)
- SMPSOC\_66V
  - 03. SMPS configuration, [184](#)
- SMPSOC\_68V
  - 03. SMPS configuration, [184](#)
- SMPSOC\_70V
  - 03. SMPS configuration, [184](#)
- SMPSOC\_72V
  - 03. SMPS configuration, [184](#)
- SMPSOC\_74V
  - 03. SMPS configuration, [184](#)
- SMPSOC\_76V
  - 03. SMPS configuration, [184](#)
- SMPSOC\_78V
  - 03. SMPS configuration, [184](#)
- SMPSOC\_80V
  - 03. SMPS configuration, [184](#)
- SMPSOVC\_ADDRESS
  - Register address map, [924](#)
- SMPSSIE\_DISABLE
  - Supply interrupt enable configuration options (0x1C), [976](#)
- SMPSSIE\_ENABLE
  - Supply interrupt enable configuration options (0x1C), [976](#)
- SOFF
  - edma\_software\_tcd\_t, [383](#)
- SPI\_ACTIVE\_HIGH

- Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [960](#)
- SPI\_ACTIVE\_LOW
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [960](#)
- SPI\_GetDefaultMasterConfig
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [961](#)
- SPI\_GetDefaultSlaveConfig
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [961](#)
- SPI\_GetStatus
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [961](#)
- SPI\_MasterDeinit
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [962](#)
- SPI\_MasterInit
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [962](#)
- SPI\_MasterSetDelay
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [962](#)
- SPI\_MasterTransfer
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [962](#)
- SPI\_MasterTransferBlocking
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [963](#)
- SPI\_SetSS
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [963](#)
- SPI\_SlaveDeinit
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [963](#)
- SPI\_SlaveInit
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [964](#)
- SPI\_SlaveTransfer
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [964](#)
- SPI\_SlaveTransferBlocking
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [964](#)
- SPI\_TRANSFER\_LSB\_FIRST
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [961](#)
- SPI\_TRANSFER\_MSB\_FIRST
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [961](#)
- SPI\_USING\_DMA
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [961](#)
- SPI\_USING\_INTERRUPTS
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [961](#)
- SPIFIE\_DISABLE
  - System interrupt enable configuration options (0x04), [984](#)
- SPIFIE\_ENABLE
  - System interrupt enable configuration options (0x04), [984](#)
- ST\_min
  - lin\_node\_attribute\_t, [759](#)
- STAT\_HSRUN
  - Power\_s32k1xx, [879](#)
- STAT\_INVALID
  - Power\_s32k1xx, [879](#)
- STAT\_RUN
  - Power\_s32k1xx, [879](#)
- STAT\_STOP
  - Power\_s32k1xx, [879](#)
- STAT\_VLPR
  - Power\_s32k1xx, [879](#)
- STAT\_VLPS
  - Power\_s32k1xx, [879](#)
- STAT\_VLPW
  - Power\_s32k1xx, [879](#)
- STCD\_ADDR
  - EDMA Driver, [383](#)
- STCD\_SIZE
  - EDMA Driver, [383](#)
- STOP\_MODE
  - Clock Manager Driver, [305](#)
- SUBFUNCTION\_NOT\_SUPPORTED
  - Common Transport Layer API, [320](#)
- SUCCESSFULL\_TRANSFER
  - Common Core API., [317](#)
- SUPC\_ADDRESS
  - Register address map, [924](#)
- SUPIE\_ADDRESS
  - Register address map, [924](#)
- SUPIS\_ADDRESS
  - Register address map, [924](#)
- SUPVS\_ADDRESS
  - Register address map, [925](#)
- SUSPEND\_WAIT\_CNT
  - Flash Memory (Flash), [425](#)
- SYS\_CLK\_MAX\_NO
  - Clock Manager Driver, [303](#)
- SYSIE\_ADDRESS
  - Register address map, [925](#)
- SYSIS\_ADDRESS
  - Register address map, [925](#)
- safeState
  - ftm\_independent\_ch\_param\_t, [580](#)
- sampleTicks
  - adc\_config\_t, [236](#)
- sampleTime
  - adc\_converter\_config\_t, [217](#)
- samples
  - cmp\_trigger\_mode\_t, [331](#)
- save\_config\_flg
  - lin\_protocol\_state\_t, [770](#)
  - lin\_word\_status\_str\_t, [756](#)
- sbc\_can\_cfdc\_t



- UJA116xA SBC Driver, [1030](#)
- sbcb\_can\_cmc\_t
  - UJA116xA SBC Driver, [1030](#)
- sbcb\_can\_conf\_t, [1018](#)
  - canConf, [1018](#)
  - canTransEvt, [1018](#)
  - datRate, [1018](#)
  - dataMask, [1018](#)
  - frame, [1018](#)
  - identif, [1019](#)
  - mask, [1019](#)
- sbcb\_can\_cpnc\_t
  - UJA116xA SBC Driver, [1030](#)
- sbcb\_can\_ctr\_t, [1016](#)
  - cfdc, [1016](#)
  - cmc, [1017](#)
  - cpnc, [1017](#)
  - pncok, [1017](#)
- sbcb\_can\_pncok\_t
  - UJA116xA SBC Driver, [1031](#)
- sbcb\_dat\_rate\_t
  - UJA116xA SBC Driver, [1031](#)
- sbcb\_data\_mask\_t
  - UJA116xA SBC Driver, [1029](#)
- sbcb\_evn\_capt\_t, [1027](#)
  - glEvt, [1027](#)
  - supEvt, [1027](#)
  - sysEvt, [1027](#)
  - transEvt, [1027](#)
  - wakePinEvt, [1027](#)
- sbcb\_factories\_conf\_t, [1021](#)
  - control, [1021](#)
  - startUp, [1021](#)
- sbcb\_fail\_safe\_lhc\_t
  - UJA116xA SBC Driver, [1031](#)
- sbcb\_fail\_safe\_rcc\_t
  - UJA116xA SBC Driver, [1029](#)
- sbcb\_frame\_ctr\_dlc\_t
  - UJA116xA SBC Driver, [1029](#)
- sbcb\_frame\_ctr\_ide\_t
  - UJA116xA SBC Driver, [1031](#)
- sbcb\_frame\_ctr\_pndm\_t
  - UJA116xA SBC Driver, [1032](#)
- sbcb\_frame\_t, [1017](#)
  - dlc, [1018](#)
  - ide, [1018](#)
  - pndm, [1018](#)
- sbcb\_gl\_evt\_stat\_supe\_t
  - UJA116xA SBC Driver, [1032](#)
- sbcb\_gl\_evt\_stat\_syse\_t
  - UJA116xA SBC Driver, [1032](#)
- sbcb\_gl\_evt\_stat\_t, [1024](#)
  - supe, [1024](#)
  - syse, [1024](#)
  - trxe, [1024](#)
  - wpe, [1024](#)
- sbcb\_gl\_evt\_stat\_trxe\_t
  - UJA116xA SBC Driver, [1032](#)
- sbcb\_gl\_evt\_stat\_wpe\_t
  - UJA116xA SBC Driver, [1032](#)
- sbcb\_identif\_mask\_t
  - UJA116xA SBC Driver, [1030](#)
- sbcb\_identifier\_t
  - UJA116xA SBC Driver, [1030](#)
- sbcb\_int\_config\_t, [1020](#)
  - can, [1020](#)
  - lhc, [1020](#)
  - lockMask, [1020](#)
  - mode, [1020](#)
  - regulatorCtr, [1020](#)
  - sysEvt, [1020](#)
  - wakePin, [1021](#)
  - watchdog, [1021](#)
- sbcb\_lock\_t
  - UJA116xA SBC Driver, [1033](#)
- sbcb\_main\_nms\_t
  - UJA116xA SBC Driver, [1033](#)
- sbcb\_main\_otws\_t
  - UJA116xA SBC Driver, [1033](#)
- sbcb\_main\_rss\_t
  - UJA116xA SBC Driver, [1033](#)
- sbcb\_main\_status\_t, [1021](#)
  - nms, [1022](#)
  - otws, [1022](#)
  - rss, [1022](#)
- sbcb\_mode\_mc\_t
  - UJA116xA SBC Driver, [1034](#)
- sbcb\_mtpnv\_stat\_eccs\_t
  - UJA116xA SBC Driver, [1034](#)
- sbcb\_mtpnv\_stat\_nvmps\_t
  - UJA116xA SBC Driver, [1034](#)
- sbcb\_mtpnv\_stat\_t, [1027](#)
  - eccs, [1028](#)
  - nvmps, [1028](#)
  - wrcnts, [1028](#)
- sbcb\_mtpnv\_stat\_wrcnts\_t
  - UJA116xA SBC Driver, [1030](#)
- sbcb\_register\_t
  - UJA116xA SBC Driver, [1034](#)
- sbcb\_regulator\_ctr\_t, [1019](#)
  - regulator, [1019](#)
  - supplyEvt, [1020](#)
- sbcb\_regulator\_pdc\_t
  - UJA116xA SBC Driver, [1036](#)
- sbcb\_regulator\_t, [1015](#)
  - pdc, [1015](#)
  - v1rtc, [1015](#)
  - v2c, [1015](#)
- sbcb\_regulator\_v1rtc\_t
  - UJA116xA SBC Driver, [1036](#)
- sbcb\_regulator\_v2c\_t
  - UJA116xA SBC Driver, [1036](#)
- sbcb\_sbc\_fnmc\_t
  - UJA116xA SBC Driver, [1036](#)
- sbcb\_sbc\_sdmc\_t
  - UJA116xA SBC Driver, [1037](#)

- sbc\_sbc\_slpc\_t
  - UJA116xA SBC Driver, [1037](#)
- sbc\_sbc\_t, [1014](#)
  - fnmc, [1014](#)
  - sdmc, [1014](#)
  - slpc, [1014](#)
  - v1rtsuc, [1014](#)
- sbc\_sbc\_v1rtsuc\_t
  - UJA116xA SBC Driver, [1037](#)
- sbc\_start\_up\_rlc\_t
  - UJA116xA SBC Driver, [1037](#)
- sbc\_start\_up\_t, [1014](#)
  - rlc, [1015](#)
  - v2suc, [1015](#)
- sbc\_start\_up\_v2suc\_t
  - UJA116xA SBC Driver, [1038](#)
- sbc\_status\_group\_t, [1028](#)
  - events, [1028](#)
  - mainS, [1028](#)
  - supply, [1028](#)
  - trans, [1028](#)
  - wakePin, [1029](#)
  - wtdog, [1029](#)
- sbc\_sup\_evnt\_stat\_t, [1025](#)
  - v1u, [1025](#)
  - v2o, [1025](#)
  - v2u, [1025](#)
- sbc\_sup\_evnt\_stat\_v1u\_t
  - UJA116xA SBC Driver, [1038](#)
- sbc\_sup\_evnt\_stat\_v2o\_t
  - UJA116xA SBC Driver, [1038](#)
- sbc\_sup\_evnt\_stat\_v2u\_t
  - UJA116xA SBC Driver, [1038](#)
- sbc\_supply\_evnt\_t, [1015](#)
  - v1ue, [1015](#)
  - v2oe, [1016](#)
  - v2ue, [1016](#)
- sbc\_supply\_evnt\_v1ue\_t
  - UJA116xA SBC Driver, [1038](#)
- sbc\_supply\_evnt\_v2oe\_t
  - UJA116xA SBC Driver, [1039](#)
- sbc\_supply\_evnt\_v2ue\_t
  - UJA116xA SBC Driver, [1039](#)
- sbc\_supply\_stat\_v1s\_t
  - UJA116xA SBC Driver, [1039](#)
- sbc\_supply\_stat\_v2s\_t
  - UJA116xA SBC Driver, [1039](#)
- sbc\_supply\_status\_t, [1022](#)
  - v1s, [1023](#)
  - v2s, [1023](#)
- sbc\_sys\_evnt\_otwe\_t
  - UJA116xA SBC Driver, [1039](#)
- sbc\_sys\_evnt\_spife\_t
  - UJA116xA SBC Driver, [1040](#)
- sbc\_sys\_evnt\_stat\_otw\_t
  - UJA116xA SBC Driver, [1040](#)
- sbc\_sys\_evnt\_stat\_po\_t
  - UJA116xA SBC Driver, [1040](#)
- sbc\_sys\_evnt\_stat\_spif\_t
  - UJA116xA SBC Driver, [1040](#)
- sbc\_sys\_evnt\_stat\_t, [1024](#)
  - otw, [1025](#)
  - po, [1025](#)
  - spif, [1025](#)
  - wdf, [1025](#)
- sbc\_sys\_evnt\_stat\_wdf\_t
  - UJA116xA SBC Driver, [1040](#)
- sbc\_sys\_evnt\_t, [1016](#)
  - owte, [1016](#)
  - spife, [1016](#)
- sbc\_trans\_evnt\_cbse\_t
  - UJA116xA SBC Driver, [1041](#)
- sbc\_trans\_evnt\_cfe\_t
  - UJA116xA SBC Driver, [1041](#)
- sbc\_trans\_evnt\_cwe\_t
  - UJA116xA SBC Driver, [1041](#)
- sbc\_trans\_evnt\_stat\_cbs\_t
  - UJA116xA SBC Driver, [1041](#)
- sbc\_trans\_evnt\_stat\_cf\_t
  - UJA116xA SBC Driver, [1042](#)
- sbc\_trans\_evnt\_stat\_cw\_t
  - UJA116xA SBC Driver, [1042](#)
- sbc\_trans\_evnt\_stat\_pnfde\_t
  - UJA116xA SBC Driver, [1042](#)
- sbc\_trans\_evnt\_stat\_t, [1026](#)
  - cbs, [1026](#)
  - cf, [1026](#)
  - cw, [1026](#)
  - pnfde, [1026](#)
- sbc\_trans\_evnt\_t, [1017](#)
  - cbse, [1017](#)
  - cfe, [1017](#)
  - cwe, [1017](#)
- sbc\_trans\_stat\_cbss\_t
  - UJA116xA SBC Driver, [1042](#)
- sbc\_trans\_stat\_cfs\_t
  - UJA116xA SBC Driver, [1042](#)
- sbc\_trans\_stat\_coscs\_t
  - UJA116xA SBC Driver, [1043](#)
- sbc\_trans\_stat\_cpnerr\_t
  - UJA116xA SBC Driver, [1043](#)
- sbc\_trans\_stat\_cpns\_t
  - UJA116xA SBC Driver, [1043](#)
- sbc\_trans\_stat\_cts\_t
  - UJA116xA SBC Driver, [1043](#)
- sbc\_trans\_stat\_t, [1023](#)
  - cbss, [1023](#)
  - cfs, [1023](#)
  - coscs, [1023](#)
  - cpnerr, [1023](#)
  - cpns, [1023](#)
  - cts, [1024](#)
  - vcs, [1024](#)
- sbc\_trans\_stat\_vcs\_t
  - UJA116xA SBC Driver, [1043](#)
- sbc\_wake\_en\_wpfe\_t



- UJA116xA SBC Driver, [1044](#)
- sbc\_wake\_en\_wpre\_t
  - UJA116xA SBC Driver, [1044](#)
- sbc\_wake\_evnt\_stat\_t, [1026](#)
  - wpf, [1026](#)
  - wpr, [1026](#)
- sbc\_wake\_evnt\_stat\_wpf\_t
  - UJA116xA SBC Driver, [1044](#)
- sbc\_wake\_evnt\_stat\_wpr\_t
  - UJA116xA SBC Driver, [1044](#)
- sbc\_wake\_stat\_wpvs\_t
  - UJA116xA SBC Driver, [1044](#)
- sbc\_wake\_t, [1019](#)
  - wpfe, [1019](#)
  - wpre, [1019](#)
- sbc\_wtdog\_ctr\_nwp\_t
  - UJA116xA SBC Driver, [1045](#)
- sbc\_wtdog\_ctr\_t, [1013](#)
  - modeControl, [1013](#)
  - nominalPeriod, [1013](#)
- sbc\_wtdog\_ctr\_wmc\_t
  - UJA116xA SBC Driver, [1045](#)
- sbc\_wtdog\_stat\_fnms\_t
  - UJA116xA SBC Driver, [1045](#)
- sbc\_wtdog\_stat\_sdms\_t
  - UJA116xA SBC Driver, [1046](#)
- sbc\_wtdog\_stat\_wds\_t
  - UJA116xA SBC Driver, [1046](#)
- sbc\_wtdog\_status\_t, [1022](#)
  - fnms, [1022](#)
  - sdms, [1022](#)
  - wds, [1022](#)
- sbcConfigCtrl
  - uja113x\_mtpnv\_cfg\_t, [208](#)
- scatterGatherEnable
  - edma\_transfer\_config\_t, [381](#)
- scatterGatherNextDescAddr
  - edma\_transfer\_config\_t, [381](#)
- scg\_async\_clock\_div\_t
  - Clock Manager Driver, [305](#)
- scg\_clock\_mode\_config\_t, [293](#)
  - alternateClock, [294](#)
  - hccrConfig, [294](#)
  - initialize, [294](#)
  - rccrConfig, [294](#)
  - vccrConfig, [294](#)
- scg\_clockout\_config\_t, [294](#)
  - initialize, [294](#)
  - source, [294](#)
- scg\_clockout\_src\_t
  - Clock Manager Driver, [305](#)
- scg\_config\_t, [294](#)
  - clockModeConfig, [295](#)
  - clockOutConfig, [295](#)
  - fircConfig, [295](#)
  - rtcConfig, [295](#)
  - sircConfig, [295](#)
  - soscConfig, [295](#)
  - spilConfig, [295](#)
- scg\_firc\_config\_t, [291](#)
  - div1, [291](#)
  - div2, [291](#)
  - enableInLowPower, [291](#)
  - enableInStop, [291](#)
  - initialize, [292](#)
  - locked, [292](#)
  - range, [292](#)
  - regulator, [292](#)
- scg\_firc\_range\_t
  - Clock Manager Driver, [305](#)
- scg\_rtc\_config\_t, [293](#)
  - initialize, [293](#)
  - rtcClkInFreq, [293](#)
- scg\_sirc\_config\_t, [290](#)
  - div1, [290](#)
  - div2, [290](#)
  - enableInLowPower, [290](#)
  - enableInStop, [291](#)
  - initialize, [291](#)
  - locked, [291](#)
  - range, [291](#)
- scg\_sirc\_range\_t
  - Clock Manager Driver, [306](#)
- scg\_sosc\_config\_t, [289](#)
  - div1, [289](#)
  - div2, [289](#)
  - enableInLowPower, [289](#)
  - enableInStop, [289](#)
  - extRef, [289](#)
  - freq, [289](#)
  - gain, [290](#)
  - initialize, [290](#)
  - locked, [290](#)
  - monitorMode, [290](#)
  - range, [290](#)
- scg\_sosc\_ext\_ref\_t
  - Clock Manager Driver, [306](#)
- scg\_sosc\_gain\_t
  - Clock Manager Driver, [306](#)
- scg\_sosc\_monitor\_mode\_t
  - Clock Manager Driver, [306](#)
- scg\_sosc\_range\_t
  - Clock Manager Driver, [306](#)
- scg\_spil\_clock\_multiply\_t
  - Clock Manager Driver, [306](#)
- scg\_spil\_clock\_prediv\_t
  - Clock Manager Driver, [307](#)
- scg\_spil\_config\_t, [292](#)
  - div1, [292](#)
  - div2, [292](#)
  - enableInStop, [292](#)
  - initialize, [293](#)
  - locked, [293](#)
  - monitorMode, [293](#)
  - mult, [293](#)
  - prediv, [293](#)

- src, [293](#)
- scg\_pll\_monitor\_mode\_t
  - Clock Manager Driver, [308](#)
- scg\_system\_clock\_config\_t, [288](#)
  - divBus, [288](#)
  - divCore, [288](#)
  - divSlow, [288](#)
  - src, [289](#)
- scg\_system\_clock\_div\_t
  - Clock Manager Driver, [308](#)
- scg\_system\_clock\_src\_t
  - Clock Manager Driver, [308](#)
- scgConfig
  - clock\_manager\_user\_config\_t, [298](#)
- sch\_tbl\_type
  - lin\_schedule\_t, [761](#)
- Schedule management, [932](#)
  - l\_sch\_set, [932](#)
  - l\_sch\_tick, [932](#)
- schedule\_start
  - lin\_protocol\_user\_config\_t, [767](#)
- schedule\_start\_entry\_ptr
  - lin\_master\_data\_t, [768](#)
- schedule\_tbl
  - lin\_protocol\_user\_config\_t, [767](#)
- sckPin
  - extension\_flexio\_for\_i2s\_t, [606](#)
  - extension\_flexio\_for\_spi\_t, [960](#)
  - flexio\_i2s\_master\_user\_config\_t, [478](#)
  - flexio\_i2s\_slave\_user\_config\_t, [479](#)
  - flexio\_spi\_master\_user\_config\_t, [496](#)
  - flexio\_spi\_slave\_user\_config\_t, [498](#)
- sclPin
  - extension\_flexio\_for\_i2c\_t, [624](#)
  - flexio\_i2c\_master\_user\_config\_t, [468](#)
- sdaPin
  - extension\_flexio\_for\_i2c\_t, [624](#)
  - flexio\_i2c\_master\_user\_config\_t, [468](#)
- sdmc
  - sbc\_sbc\_t, [1014](#)
- sdms
  - sbc\_wdog\_status\_t, [1022](#)
- secondChannelPolarity
  - ftm\_combined\_ch\_param\_t, [581](#)
  - ftm\_independent\_ch\_param\_t, [580](#)
- secondChannelSafeState
  - ftm\_combined\_ch\_param\_t, [581](#)
- secondEdge
  - ftm\_combined\_ch\_param\_t, [581](#)
- secondIntConfig
  - rtc\_seconds\_int\_config\_t, [898](#)
- secondIntEnable
  - rtc\_seconds\_int\_config\_t, [898](#)
- seconds
  - rtc\_timedate\_t, [895](#)
- secondsCallbackParams
  - rtc\_seconds\_int\_config\_t, [898](#)
- sectorEraseCount
  - Flash Memory (Flash), [436](#)
- Security PAL, [933](#)
  - SECURITY\_BOOT\_MAC, [937](#)
  - SECURITY\_BOOT\_MAC\_KEY, [937](#)
  - SECURITY\_BOOT\_NOT\_DEFINED, [936](#)
  - SECURITY\_BOOT\_PARALLEL, [935](#)
  - SECURITY\_BOOT\_SERIAL, [935](#)
  - SECURITY\_BOOT\_STRICT, [935](#)
  - SECURITY\_BootDefine, [937](#)
  - SECURITY\_BootFailure, [937](#)
  - SECURITY\_BootOk, [938](#)
  - SECURITY\_CMD\_BOOT\_DEFINE, [936](#)
  - SECURITY\_CMD\_BOOT\_FAILURE, [936](#)
  - SECURITY\_CMD\_BOOT\_OK, [936](#)
  - SECURITY\_CMD\_DBG\_AUTH, [936](#)
  - SECURITY\_CMD\_DBG\_CHAL, [936](#)
  - SECURITY\_CMD\_DEC\_CBC, [936](#)
  - SECURITY\_CMD\_DEC\_ECB, [936](#)
  - SECURITY\_CMD\_ENC\_CBC, [936](#)
  - SECURITY\_CMD\_ENC\_ECB, [936](#)
  - SECURITY\_CMD\_EXPORT\_RAM\_KEY, [936](#)
  - SECURITY\_CMD\_EXTEND\_SEED, [936](#)
  - SECURITY\_CMD\_GENERATE\_MAC, [936](#)
  - SECURITY\_CMD\_GET\_ID, [936](#)
  - SECURITY\_CMD\_INIT\_RNG, [936](#)
  - SECURITY\_CMD\_LOAD\_KEY, [936](#)
  - SECURITY\_CMD\_LOAD\_PLAIN\_KEY, [936](#)
  - SECURITY\_CMD\_MP\_COMPRESS, [936](#)
  - SECURITY\_CMD\_RESERVED\_1, [936](#)
  - SECURITY\_CMD\_RESERVED\_2, [936](#)
  - SECURITY\_CMD\_RESERVED\_3, [936](#)
  - SECURITY\_CMD\_RND, [936](#)
  - SECURITY\_CMD\_VERIFY\_MAC, [936](#)
  - SECURITY\_CancelCommand, [938](#)
  - SECURITY\_DbgAuth, [938](#)
  - SECURITY\_DbgChal, [938](#)
  - SECURITY\_DecryptCbc, [939](#)
  - SECURITY\_DecryptCbcBlocking, [939](#)
  - SECURITY\_DecryptEcb, [940](#)
  - SECURITY\_DecryptEcbBlocking, [940](#)
  - SECURITY\_Deinit, [940](#)
  - SECURITY\_EncryptCbc, [941](#)
  - SECURITY\_EncryptCbcBlocking, [941](#)
  - SECURITY\_EncryptEcb, [941](#)
  - SECURITY\_EncryptEcbBlocking, [942](#)
  - SECURITY\_ExportRamKey, [942](#)
  - SECURITY\_ExtendSeed, [943](#)
  - SECURITY\_GenerateMac, [943](#)
  - SECURITY\_GenerateMacBlocking, [943](#)
  - SECURITY\_GenerateRnd, [945](#)
  - SECURITY\_GenerateTrnd, [945](#)
  - SECURITY\_GetAsyncCmdStatus, [945](#)
  - SECURITY\_GetDefaultConfig, [946](#)
  - SECURITY\_GetId, [946](#)
  - SECURITY\_INSTANCE0, [936](#)
  - SECURITY\_Init, [946](#)
  - SECURITY\_InitRng, [946](#)
  - SECURITY\_KEY\_1, [937](#)

- SECURITY\_KEY\_10, [937](#)
- SECURITY\_KEY\_11, [937](#)
- SECURITY\_KEY\_12, [937](#)
- SECURITY\_KEY\_13, [937](#)
- SECURITY\_KEY\_14, [937](#)
- SECURITY\_KEY\_15, [937](#)
- SECURITY\_KEY\_16, [937](#)
- SECURITY\_KEY\_17, [937](#)
- SECURITY\_KEY\_2, [937](#)
- SECURITY\_KEY\_3, [937](#)
- SECURITY\_KEY\_4, [937](#)
- SECURITY\_KEY\_5, [937](#)
- SECURITY\_KEY\_6, [937](#)
- SECURITY\_KEY\_7, [937](#)
- SECURITY\_KEY\_8, [937](#)
- SECURITY\_KEY\_9, [937](#)
- SECURITY\_LoadKey, [948](#)
- SECURITY\_LoadPlainKey, [948](#)
- SECURITY\_MASTER\_ECU, [936](#)
- SECURITY\_MPCompress, [948](#)
- SECURITY\_RAM\_KEY, [937](#)
- SECURITY\_SECRET\_KEY, [936](#)
- SECURITY\_SecureBoot, [949](#)
- SECURITY\_VerifyMac, [949](#)
- SECURITY\_VerifyMacBlocking, [950](#)
- security\_boot\_flavor\_t, [935](#)
- security\_cmd\_t, [936](#)
- security\_instance\_t, [936](#)
- security\_key\_id\_t, [936](#)
- Security Peripheral Abstraction Layer - SECURITY PAL, [951](#)
- security\_boot\_flavor\_t
  - Security PAL, [935](#)
- security\_cmd\_t
  - Security PAL, [936](#)
- security\_instance\_t
  - Security PAL, [936](#)
- security\_key\_id\_t
  - Security PAL, [936](#)
- security\_user\_config\_t, [935](#)
  - callback, [935](#)
  - callbackParam, [935](#)
- seed
  - crc\_user\_config\_t, [251](#)
- send\_functional\_request\_flg
  - lin\_master\_data\_t, [769](#)
- send\_slave\_res\_flg
  - lin\_master\_data\_t, [769](#)
- seq
  - csec\_state\_t, [262](#)
- seqErrIntEnable
  - pdb\_timer\_config\_t, [849](#)
- Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [954](#)
  - READ\_ON\_EVEN\_EDGE, [960](#)
  - READ\_ON\_ODD\_EDGE, [960](#)
  - SPI\_ACTIVE\_HIGH, [960](#)
  - SPI\_ACTIVE\_LOW, [960](#)
  - SPI\_GetDefaultMasterConfig, [961](#)
  - SPI\_GetDefaultSlaveConfig, [961](#)
  - SPI\_GetStatus, [961](#)
  - SPI\_MasterDeinit, [962](#)
  - SPI\_MasterInit, [962](#)
  - SPI\_MasterSetDelay, [962](#)
  - SPI\_MasterTransfer, [962](#)
  - SPI\_MasterTransferBlocking, [963](#)
  - SPI\_SetSS, [963](#)
  - SPI\_SlaveDeinit, [963](#)
  - SPI\_SlaveInit, [964](#)
  - SPI\_SlaveTransfer, [964](#)
  - SPI\_SlaveTransferBlocking, [964](#)
  - SPI\_TRANSFER\_LSB\_FIRST, [961](#)
  - SPI\_TRANSFER\_MSB\_FIRST, [961](#)
  - SPI\_USING\_DMA, [961](#)
  - SPI\_USING\_INTERRUPTS, [961](#)
  - spi\_clock\_phase\_t, [960](#)
  - spi\_polarity\_t, [960](#)
  - spi\_transfer\_bit\_order\_t, [960](#)
  - spi\_transfer\_type\_t, [961](#)
- serial\_0
  - lin\_serial\_number\_t, [757](#)
- serial\_1
  - lin\_serial\_number\_t, [757](#)
- serial\_2
  - lin\_serial\_number\_t, [757](#)
- serial\_3
  - lin\_serial\_number\_t, [757](#)
- serial\_number
  - lin\_node\_attribute\_t, [759](#)
- service\_flags\_ptr
  - lin\_node\_attribute\_t, [759](#)
- service\_status
  - lin\_tl\_descriptor\_t, [765](#)
- service\_supported\_ptr
  - lin\_node\_attribute\_t, [759](#)
- Signal interaction, [966](#)
- sim\_clkout\_div\_t
  - Clock Manager Driver, [308](#)
- sim\_clkout\_src\_t
  - Clock Manager Driver, [309](#)
- sim\_clock\_config\_t, [287](#)
  - clockOutConfig, [288](#)
  - lpoClockConfig, [288](#)
  - platGateConfig, [288](#)
  - qspiRefClkGating, [288](#)
  - tclkConfig, [288](#)
  - traceClockConfig, [288](#)
- sim\_clock\_out\_config\_t, [284](#)
  - divider, [284](#)
  - enable, [284](#)
  - initialize, [284](#)
  - source, [284](#)
- sim\_lpo\_clock\_config\_t, [284](#)
  - enableLpo1k, [285](#)
  - enableLpo32k, [285](#)
  - initialize, [285](#)

- sourceLpoClk, 285
- sourceRtcClk, 285
- sim\_lpoclk\_sel\_src\_t
  - Clock Manager Driver, 309
- sim\_plat\_gate\_config\_t, 285
  - enableDma, 286
  - enableEim, 286
  - enableErm, 286
  - enableMpu, 286
  - enableMscm, 286
  - initialize, 286
- sim\_qspi\_ref\_clk\_gating\_t, 286
  - enableQspiRefClk, 286
- sim\_rtc\_clk\_sel\_src\_t
  - Clock Manager Driver, 309
- sim\_tclk\_config\_t, 285
  - extPinSrc, 285
  - initialize, 285
  - tclkFreq, 285
- sim\_trace\_clock\_config\_t, 287
  - divEnable, 287
  - divFraction, 287
  - divider, 287
  - initialize, 287
  - source, 287
- simConfig
  - clock\_manager\_user\_config\_t, 298
- sircConfig
  - scg\_config\_t, 295
- slave\_ifc\_handle
  - lin\_protocol\_user\_config\_t, 767
- slave\_resp\_cnt
  - lin\_tl\_descriptor\_t, 765
- slaveAddress
  - flexio\_i2c\_master\_user\_config\_t, 469
  - i2c\_master\_t, 625
  - i2c\_slave\_t, 626
  - lpi2c\_master\_user\_config\_t, 674
  - lpi2c\_slave\_user\_config\_t, 675
- slaveCallback
  - lpi2c\_slave\_user\_config\_t, 675
- slaveListening
  - i2c\_slave\_t, 626
  - lpi2c\_slave\_user\_config\_t, 675
- sleepOnExitValue
  - power\_manager\_user\_config\_t, 878
- slpc
  - sbc\_sbc\_t, 1014
- smc\_power\_mode\_config\_t, 878
  - powerModeName, 878
- smc\_power\_mode\_protection\_config\_t, 878
  - vlProt, 878
- smc\_run\_mode\_t
  - Power\_s32k1xx, 880
- smc\_stop\_mode\_t
  - Power\_s32k1xx, 880
- smc\_stop\_option\_t
  - Power\_s32k1xx, 880
- smpsCtrl
  - uja113x\_smpsc\_cfg\_t, 183
- smpsOutVolt
  - uja113x\_smpsc\_cfg\_t, 183
- smpsOvertempCtrl
  - uja113x\_smpsc\_cfg\_t, 183
- smpscSmpsCfg
  - uja113x\_init\_cfg\_t, 146
- SoC Header file (SoC Header ), 967
- SoC Support, 968
- softwareSync
  - ftm\_pwm\_sync\_t, 524
- soscConfig
  - scg\_config\_t, 295
- source
  - module\_clk\_config\_t, 298
  - scg\_clockout\_config\_t, 294
  - sim\_clock\_out\_config\_t, 284
  - sim\_trace\_clock\_config\_t, 287
- sourceClock
  - pwm\_ftm\_timebase\_t, 887
- sourceLpoClk
  - sim\_lpo\_clock\_config\_t, 285
- sourceRtcClk
  - sim\_lpo\_clock\_config\_t, 285
- spi\_clock\_phase\_t
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), 960
- spi\_instance\_t, 1088
  - instIdx, 1088
  - instType, 1088
- spi\_master\_t, 957
  - baudRate, 957
  - bitOrder, 957
  - callback, 957
  - callbackParam, 957
  - clockPhase, 957
  - clockPolarity, 957
  - extension, 957
  - frameSize, 958
  - rxDMACHannel, 958
  - ssPin, 958
  - ssPolarity, 958
  - transferType, 958
  - txDMACHannel, 958
- spi\_polarity\_t
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), 960
- spi\_slave\_t, 958
  - bitOrder, 958
  - callback, 959
  - callbackParam, 959
  - clockPhase, 959
  - clockPolarity, 959
  - extension, 959
  - frameSize, 959
  - rxDMACHannel, 959
  - ssPolarity, 959

- transferType, [959](#)
- txDMAChannel, [959](#)
- spi\_transfer\_bit\_order\_t
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [960](#)
- spi\_transfer\_type\_t
  - Serial Peripheral Interface - Peripheral Abstraction Layer(SPI PAL), [961](#)
- spif
  - sbc\_sys\_evnt\_stat\_t, [1025](#)
- spife
  - sbc\_sys\_evnt\_t, [1016](#)
- spilConfig
  - scg\_config\_t, [295](#)
- src
  - scg\_spil\_config\_t, [293](#)
  - scg\_system\_clock\_config\_t, [289](#)
  - sys\_clk\_config\_t, [299](#)
- srcAddr
  - edma\_transfer\_config\_t, [381](#)
- srcLastAddrAdjust
  - edma\_transfer\_config\_t, [381](#)
- srcModulo
  - edma\_transfer\_config\_t, [382](#)
- srcOffset
  - edma\_transfer\_config\_t, [382](#)
- srcOffsetEnable
  - edma\_loop\_transfer\_config\_t, [380](#)
- srcTransferSize
  - edma\_transfer\_config\_t, [382](#)
- ssPin
  - extension\_flexio\_for\_spi\_t, [960](#)
  - flexio\_spi\_master\_user\_config\_t, [496](#)
  - flexio\_spi\_slave\_user\_config\_t, [498](#)
  - spi\_master\_t, [958](#)
- ssPolarity
  - spi\_master\_t, [958](#)
  - spi\_slave\_t, [959](#)
- Start-up control configuration options (0x73), [970](#)
  - IO2SFC\_LIMP\_HOME, [971](#)
  - IO2SFC\_STD\_IO, [971](#)
  - IO3SFC\_LIMP\_HOME, [971](#)
  - IO3SFC\_STD\_IO, [971](#)
  - IO4SFC\_LIMP\_HOME, [971](#)
  - IO4SFC\_STD\_IO, [971](#)
  - RLC\_PULSE\_WIDTH\_01\_MS, [971](#)
  - RLC\_PULSE\_WIDTH\_03\_05\_MS, [971](#)
  - RLC\_PULSE\_WIDTH\_10\_12\_MS, [971](#)
  - RLC\_PULSE\_WIDTH\_20\_25\_MS, [971](#)
  - V2SUC\_ALWAYS\_OFF, [972](#)
  - V2SUC\_ALWAYS\_ON, [972](#)
- startAddr
  - mpu\_region\_config\_t, [803](#)
  - mpu\_user\_config\_t, [791](#)
- startUp
  - sbc\_factories\_conf\_t, [1021](#)
- startupCtrl
  - uja113x\_mtpnv\_cfg\_t, [208](#)
- state
  - cmp\_dac\_t, [330](#)
  - flexcan\_mb\_handle\_t, [448](#)
- staticCallbacks
  - power\_manager\_state\_t, [868](#)
- staticCallbacksNumber
  - power\_manager\_state\_t, [868](#)
- status
  - edma\_chn\_state\_t, [377](#)
  - lpspi\_state\_t, [707](#)
- statusRegisterLock
  - rtc\_register\_lock\_config\_t, [899](#)
- stop
  - wdg\_option\_mode\_t, [1062](#)
  - wdog\_op\_mode\_t, [1071](#)
- stopBitCount
  - lpuart\_user\_config\_t, [735](#)
  - uart\_user\_config\_t, [1053](#)
- Structural Core Self Test, [973](#)
- successful\_transfer
  - lin\_protocol\_state\_t, [770](#)
  - lin\_word\_status\_str\_t, [756](#)
- supEvt
  - sbc\_evn\_capt\_t, [1027](#)
- supe
  - sbc\_gl\_evnt\_stat\_t, [1024](#)
- supieMask
  - uja113x\_init\_cfg\_t, [146](#)
- supplier\_id
  - lin\_product\_id\_t, [1085](#)
- supply
  - sbc\_status\_group\_t, [1028](#)
- Supply interrupt enable configuration options (0x1C), [975](#)
  - BMOIE\_DISABLE, [976](#)
  - BMOIE\_ENABLE, [976](#)
  - BMUIE\_DISABLE, [976](#)
  - BMUIE\_ENABLE, [976](#)
  - SMPSSIE\_DISABLE, [976](#)
  - SMPSSIE\_ENABLE, [976](#)
  - V1UIE\_DISABLE, [976](#)
  - V1UIE\_ENABLE, [976](#)
  - VEXTIOE\_DISABLE, [976](#)
  - VEXTIOE\_ENABLE, [976](#)
  - VEXTUIE\_DISABLE, [977](#)
  - VEXTUIE\_ENABLE, [977](#)
- supplyEvt
  - sbc\_regulator\_ctr\_t, [1020](#)
- supplyMonitoringEnable
  - adc\_converter\_config\_t, [217](#)
  - extension\_adc\_s32k1xx\_t, [237](#)
- syncMethod
  - ftm\_user\_config\_t, [525](#)
- syncPoint
  - ftm\_pwm\_sync\_t, [524](#)
- sys\_clk\_config\_t, [298](#)
- dividers, [299](#)
- src, [299](#)

- sysEvt
  - sbc\_evn\_capt\_t, [1027](#)
  - sbc\_int\_config\_t, [1020](#)
- syse
  - sbc\_gl\_evt\_stat\_t, [1024](#)
- sysieMask
  - uja113x\_init\_cfg\_t, [146](#)
- System Basis Chip Driver (SBC) - UJA116xA Family, [978](#)
- System Basis Chip library (SBC) - UJA113x Family, [983](#)
- System interrupt enable configuration options (0x04), [984](#)
  - OTWIE\_DISABLE, [984](#)
  - OTWIE\_ENABLE, [984](#)
  - SPIFIE\_DISABLE, [984](#)
  - SPIFIE\_ENABLE, [984](#)
- TIE\_ADDRESS
  - Register address map, [925](#)
- TIM1C\_ADDRESS
  - Register address map, [925](#)
- TIM1DCC\_ADDRESS
  - Register address map, [925](#)
- TIM2C\_ADDRESS
  - Register address map, [925](#)
- TIM2DCC\_ADDRESS
  - Register address map, [925](#)
- TIM3C\_ADDRESS
  - Register address map, [925](#)
- TIM3DCC\_ADDRESS
  - Register address map, [925](#)
- TIM4C\_ADDRESS
  - Register address map, [925](#)
- TIM4DCC\_ADDRESS
  - Register address map, [926](#)
- TIMER\_CHAN\_TYPE\_CONTINUOUS
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [999](#)
- TIMER\_CHAN\_TYPE\_ONESHOT
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [999](#)
- TIMER\_RESOLUTION\_TYPE\_MICROSECOND
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [999](#)
- TIMER\_RESOLUTION\_TYPE\_MILLISECOND
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [999](#)
- TIMER\_RESOLUTION\_TYPE\_NANOSECOND
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [999](#)
- TIMING\_Deinit
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [999](#)
- TIMING\_DisableNotification
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [999](#)
- TIMING\_EnableNotification
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [999](#)
- TIMING\_GetElapsed
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [1000](#)
- TIMING\_GetMaxPeriod
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [1000](#)
- TIMING\_GetRemaining
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [1000](#)
- TIMING\_GetResolution
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [1001](#)
- TIMING\_Init
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [1001](#)
- TIMING\_InstallCallback
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [1001](#)
- TIMING\_StartChannel
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [1002](#)
- TIMING\_StopChannel
  - Timing - Peripheral Abstraction Layer (TIMING P↔AL), [1002](#)
- TIS\_ADDRESS
  - Register address map, [926](#)
- TL\_ACTION\_ID\_IGNORE
  - Low level API, [777](#)
- TL\_ACTION\_NONE
  - Low level API, [777](#)
- TL\_ERROR
  - Low level API, [777](#)
- TL\_HANDLER\_INTERLEAVE\_MODE
  - Low level API, [777](#)
- TL\_MAKE\_RES\_DATA
  - Low level API, [777](#)
- TL\_RECEIVE\_MESSAGE
  - Low level API, [777](#)
- TL\_RX\_COMPLETED
  - Low level API, [777](#)
- TL\_SLAVE\_GET\_ACTION
  - Low level API, [777](#)
- TL\_TIMEOUT\_SERVICE
  - Low level API, [777](#)
- TL\_TX\_COMPLETED
  - Low level API, [777](#)
- TRANSMITTING
  - Common Transport Layer API, [320](#)
- TRGMUX Driver, [985](#)
  - TRGMUX\_DRV\_Deinit, [988](#)
  - TRGMUX\_DRV\_GenSWTrigger, [988](#)
  - TRGMUX\_DRV\_GetLockForTargetModule, [988](#)
  - TRGMUX\_DRV\_GetTrigSourceForTargetModule, [988](#)
  - TRGMUX\_DRV\_Init, [989](#)
  - TRGMUX\_DRV\_SetLockForTargetModule, [989](#)
  - TRGMUX\_DRV\_SetTrigSourceForTargetModule, [989](#)
  - trgmux\_target\_module\_t, [987](#)



- trgmux\_trigger\_source\_t, [987](#)
- TRGMUX\_DRV\_Deinit
  - TRGMUX Driver, [988](#)
- TRGMUX\_DRV\_GenSWTrigger
  - TRGMUX Driver, [988](#)
- TRGMUX\_DRV\_GetLockForTargetModule
  - TRGMUX Driver, [988](#)
- TRGMUX\_DRV\_GetTrigSourceForTargetModule
  - TRGMUX Driver, [988](#)
- TRGMUX\_DRV\_Init
  - TRGMUX Driver, [989](#)
- TRGMUX\_DRV\_SetLockForTargetModule
  - TRGMUX Driver, [989](#)
- TRGMUX\_DRV\_SetTrigSourceForTargetModule
  - TRGMUX Driver, [989](#)
- TS\_ADDRESS
  - Register address map, [926](#)
- TXMC\_PWM
  - 05. Timer control, [198](#)
- TXMC\_TIM
  - 05. Timer control, [198](#)
- TXMC\_TRIG\_END
  - 05. Timer control, [198](#)
- TXMC\_TRIG\_START
  - 05. Timer control, [198](#)
- TXPC\_1000MS\_PERIOD
  - 05. Timer control, [198](#)
- TXPC\_100MS\_PERIOD
  - 05. Timer control, [198](#)
- TXPC\_2000MS\_PERIOD
  - 05. Timer control, [198](#)
- TXPC\_200MS\_PERIOD
  - 05. Timer control, [198](#)
- TXPC\_20MS\_PERIOD
  - 05. Timer control, [198](#)
- TXPC\_30MS\_PERIOD
  - 05. Timer control, [198](#)
- TXPC\_4000MS\_PERIOD
  - 05. Timer control, [198](#)
- TXPC\_400MS\_PERIOD
  - 05. Timer control, [198](#)
- TXPC\_4MS\_PERIOD
  - 05. Timer control, [198](#)
- TXPC\_50MS\_PERIOD
  - 05. Timer control, [198](#)
- TXPC\_800MS\_PERIOD
  - 05. Timer control, [198](#)
- TXPC\_8MS\_PERIOD
  - 05. Timer control, [198](#)
- targetClockConfigIndex
  - clock\_notify\_struct\_t, [300](#)
- targetModule
  - trgmux\_inout\_mapping\_config\_t, [986](#)
- targetPowerConfigIndex
  - power\_manager\_notify\_struct\_t, [867](#)
- targetPowerConfigPtr
  - power\_manager\_notify\_struct\_t, [867](#)
- tclkConfig
  - sim\_clock\_config\_t, [288](#)
- tclkFreq
  - sim\_tclk\_config\_t, [285](#)
- tieMask
  - uja113x\_init\_cfg\_t, [146](#)
- timXDutyCycleCtrl
  - uja113x\_timsc\_cfg\_t, [197](#)
- timXModeCtrl
  - uja113x\_timsc\_cfg\_t, [197](#)
- timXPeriodCtrl
  - uja113x\_timsc\_cfg\_t, [197](#)
- timeCompensationRegisterLock
  - rtc\_register\_lock\_config\_t, [899](#)
- timeInvalidIntEnable
  - rtc\_interrupt\_config\_t, [898](#)
- timebase
  - pwm\_channel\_t, [888](#)
- timeout
  - uja113x\_driver\_t, [147](#)
  - uja113x\_drv\_cfg\_t, [143](#)
- timeoutCounter
  - lin\_state\_t, [656](#)
- timeoutCounterFlag
  - lin\_state\_t, [656](#)
- timeoutValue
  - wdg\_config\_t, [1063](#)
  - wdog\_user\_config\_t, [1072](#)
- timer\_chan\_config\_t, [996](#)
  - callback, [997](#)
  - callbackParam, [997](#)
  - chanType, [997](#)
  - channel, [997](#)
- timer\_chan\_state\_t, [1088](#)
- timer\_chan\_type\_t
  - Timing - Peripheral Abstraction Layer (TIMING PAL), [999](#)
- timer\_config\_t, [997](#)
  - chanConfigArray, [997](#)
  - extension, [997](#)
  - numChan, [997](#)
- timer\_resolution\_type\_t
  - Timing - Peripheral Abstraction Layer (TIMING PAL), [999](#)
- timerGetTimeIntervalCallback
  - lin\_user\_config\_t, [654](#)
- timerGetTimeIntervalCallbackArr
  - Low level API, [783](#)
- timerMode
  - lpit\_user\_channel\_config\_t, [691](#)
- Timing - Peripheral Abstraction Layer (TIMING PAL), [992](#)
  - TIMER\_CHAN\_TYPE\_CONTINUOUS, [999](#)
  - TIMER\_CHAN\_TYPE\_ONESHOT, [999](#)
  - TIMER\_RESOLUTION\_TYPE\_MICROSECOND, [999](#)
  - TIMER\_RESOLUTION\_TYPE\_MILLISECOND, [999](#)
  - TIMER\_RESOLUTION\_TYPE\_NANOSECOND, [999](#)

- TIMING\_Deinit, 999
- TIMING\_DisableNotification, 999
- TIMING\_EnableNotification, 999
- TIMING\_GetElapsed, 1000
- TIMING\_GetMaxPeriod, 1000
- TIMING\_GetRemaining, 1000
- TIMING\_GetResolution, 1001
- TIMING\_Init, 1001
- TIMING\_InstallCallback, 1001
- TIMING\_StartChannel, 1002
- TIMING\_StopChannel, 1002
- timer\_chan\_type\_t, 999
- timer\_resolution\_type\_t, 999
- timing\_instance\_t, 1089
  - instIdx, 1089
  - instType, 1089
- timscTimer1Cfg
  - uja113x\_init\_cfg\_t, 146
- timscTimer2Cfg
  - uja113x\_init\_cfg\_t, 147
- timscTimer3Cfg
  - uja113x\_init\_cfg\_t, 147
- timscTimer4Cfg
  - uja113x\_init\_cfg\_t, 147
- tl\_pdu\_ptr
  - lin\_transport\_layer\_queue\_t, 762
- tl\_queue\_data
  - lin\_schedule\_data\_t, 761
- tl\_rx\_queue
  - lin\_tl\_descriptor\_t, 765
- tl\_rx\_queue\_data\_ptr
  - lin\_protocol\_user\_config\_t, 767
- tl\_tx\_queue
  - lin\_tl\_descriptor\_t, 765
- tl\_tx\_queue\_data\_ptr
  - lin\_protocol\_user\_config\_t, 767
- traceClockConfig
  - sim\_clock\_config\_t, 288
- trans
  - sbc\_status\_group\_t, 1028
- transEvt
  - sbc\_evn\_capt\_t, 1027
- Transceiver interrupt enable configuration options
  - (0x23), 1003
  - CBSIE\_DISABLE, 1004
  - CBSIE\_ENABLE, 1004
  - CFIE\_DISABLE, 1004
  - CFIE\_ENABLE, 1004
  - CWIE\_DISABLE, 1004
  - CWIE\_ENABLE, 1004
  - L1WIE\_DISABLE, 1004
  - L1WIE\_ENABLE, 1004
  - L2WIE\_DISABLE, 1004
  - L2WIE\_ENABLE, 1004
- transfer\_status\_t
  - LPSPi Driver, 710
- transfer\_type
  - flexcan\_user\_config\_t, 451
- transferSize
  - flexio\_spi\_master\_user\_config\_t, 496
  - flexio\_spi\_slave\_user\_config\_t, 498
- transferType
  - FlexCANState, 448
  - i2c\_master\_t, 625
  - i2c\_slave\_t, 626
  - i2s\_user\_config\_t, 605
  - lpi2c\_master\_user\_config\_t, 674
  - lpi2c\_slave\_user\_config\_t, 675
  - lpspi\_master\_config\_t, 704
  - lpspi\_slave\_config\_t, 708
  - lpspi\_state\_t, 707
  - lpuart\_state\_t, 734
  - lpuart\_user\_config\_t, 735
  - spi\_master\_t, 958
  - spi\_slave\_t, 959
  - uart\_user\_config\_t, 1053
- transmit\_error\_resp\_sig\_flg
  - lin\_protocol\_state\_t, 771
- transmitStatus
  - lpuart\_state\_t, 734
- Transport layer API, 1005
- trgmux\_inout\_mapping\_config\_t, 986
  - lockTargetModuleReg, 986
  - targetModule, 986
  - triggerSource, 987
- trgmux\_target\_module\_t
  - TRGMUX Driver, 987
- trgmux\_trigger\_source\_t
  - TRGMUX Driver, 987
- trgmux\_user\_config\_t, 987
  - inOutMappingConfig, 987
  - numInOutMappingConfigs, 987
- trigger
  - adc\_converter\_config\_t, 217
- triggerInput
  - pdb\_timer\_config\_t, 849
- triggerMode
  - cmp\_module\_t, 332
- triggerSel
  - adc\_converter\_config\_t, 217
- triggerSelect
  - lpit\_user\_channel\_config\_t, 691
- triggerSource
  - adc\_group\_config\_t, 235
  - lpit\_user\_channel\_config\_t, 691
  - trgmux\_inout\_mapping\_config\_t, 987
- trimValue
  - pmc\_lpo\_clock\_config\_t, 297
- trxe
  - sbc\_gl\_evt\_stat\_t, 1024
- tx\_msg\_size
  - lin\_tl\_descriptor\_t, 765
- tx\_msg\_status
  - lin\_tl\_descriptor\_t, 765
- txBuff
  - lin\_state\_t, 656



- lpspi\_state\_t, [707](#)
- lpuart\_state\_t, [734](#)
- txCallback
  - lpuart\_state\_t, [734](#)
  - uart\_user\_config\_t, [1053](#)
- txCallbackParam
  - lpuart\_state\_t, [734](#)
  - uart\_user\_config\_t, [1053](#)
- txComplete
  - lpuart\_state\_t, [734](#)
- txCompleted
  - lin\_state\_t, [656](#)
- txCount
  - lpspi\_state\_t, [707](#)
- txDMACHannel
  - flexio\_i2c\_master\_user\_config\_t, [469](#)
  - flexio\_i2s\_master\_user\_config\_t, [478](#)
  - flexio\_i2s\_slave\_user\_config\_t, [480](#)
  - flexio\_spi\_master\_user\_config\_t, [496](#)
  - flexio\_spi\_slave\_user\_config\_t, [498](#)
  - i2s\_user\_config\_t, [605](#)
  - lpspi\_master\_config\_t, [704](#)
  - lpspi\_slave\_config\_t, [708](#)
  - lpspi\_state\_t, [707](#)
  - lpuart\_user\_config\_t, [735](#)
  - spi\_master\_t, [958](#)
  - spi\_slave\_t, [959](#)
  - uart\_user\_config\_t, [1053](#)
- txFrameCnt
  - lpspi\_state\_t, [707](#)
- txPin
  - extension\_flexio\_for\_i2s\_t, [606](#)
  - flexio\_i2s\_master\_user\_config\_t, [478](#)
  - flexio\_i2s\_slave\_user\_config\_t, [480](#)
- txSize
  - lin\_state\_t, [656](#)
  - lpuart\_state\_t, [734](#)
- type
  - edma\_scatter\_gather\_list\_t, [378](#)
- UART\_10\_BITS\_PER\_CHAR
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)
- UART\_15\_BITS\_PER\_CHAR
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)
- UART\_16\_BITS\_PER\_CHAR
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)
- UART\_7\_BITS\_PER\_CHAR
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)
- UART\_8\_BITS\_PER\_CHAR
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)
- UART\_9\_BITS\_PER\_CHAR
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)
- UART\_AbortReceivingData
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)
- UART\_AbortSendingData
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1055](#)
- UART\_Deinit
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1055](#)
- UART\_GetBaudRate
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1055](#)
- UART\_GetDefaultConfig
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1055](#)
- UART\_GetReceiveStatus
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1056](#)
- UART\_GetTransmitStatus
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1056](#)
- UART\_Init
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1057](#)
- UART\_ONE\_STOP\_BIT
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)
- UART\_PARITY\_DISABLED
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)
- UART\_PARITY\_EVEN
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)
- UART\_PARITY\_ODD
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)
- UART\_ReceiveData
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1057](#)
- UART\_ReceiveDataBlocking
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1057](#)
- UART\_SendData
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1058](#)
- UART\_SendDataBlocking
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1058](#)
- UART\_SetBaudRate
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1058](#)
- UART\_SetRxBuffer
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1059](#)
- UART\_SetTxBuffer
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1059](#)
- UART\_TWO\_STOP\_BIT
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)



SBC\_UJA\_CAN\_PNCOK\_DIS, [1031](#)  
SBC\_UJA\_CAN\_PNCOK\_EN, [1031](#)  
SBC\_UJA\_COUNT\_DMASK, [1029](#)  
SBC\_UJA\_COUNT\_ID\_REG, [1029](#)  
SBC\_UJA\_COUNT\_MASK, [1029](#)  
SBC\_UJA\_DAT\_MASK\_0, [1035](#)  
SBC\_UJA\_DAT\_MASK\_1, [1035](#)  
SBC\_UJA\_DAT\_MASK\_2, [1035](#)  
SBC\_UJA\_DAT\_MASK\_3, [1035](#)  
SBC\_UJA\_DAT\_MASK\_4, [1035](#)  
SBC\_UJA\_DAT\_MASK\_5, [1035](#)  
SBC\_UJA\_DAT\_MASK\_6, [1035](#)  
SBC\_UJA\_DAT\_MASK\_7, [1035](#)  
SBC\_UJA\_DAT\_RATE, [1035](#)  
SBC\_UJA\_DAT\_RATE\_CDR\_1000KB, [1031](#)  
SBC\_UJA\_DAT\_RATE\_CDR\_100KB, [1031](#)  
SBC\_UJA\_DAT\_RATE\_CDR\_125KB, [1031](#)  
SBC\_UJA\_DAT\_RATE\_CDR\_250KB, [1031](#)  
SBC\_UJA\_DAT\_RATE\_CDR\_500KB, [1031](#)  
SBC\_UJA\_DAT\_RATE\_CDR\_50KB, [1031](#)  
SBC\_UJA\_FAIL\_SAFE, [1035](#)  
SBC\_UJA\_FAIL\_SAFE\_LHC\_FLOAT, [1031](#)  
SBC\_UJA\_FAIL\_SAFE\_LHC\_LOW, [1031](#)  
SBC\_UJA\_FRAME\_CTR, [1035](#)  
SBC\_UJA\_FRAME\_CTR\_IDE\_11B, [1032](#)  
SBC\_UJA\_FRAME\_CTR\_IDE\_29B, [1032](#)  
SBC\_UJA\_FRAME\_CTR\_PNDM\_DCARE, [1032](#)  
SBC\_UJA\_FRAME\_CTR\_PNDM\_EVAL, [1032](#)  
SBC\_UJA\_GL\_EVNT\_STAT, [1035](#)  
SBC\_UJA\_GL\_EVNT\_STAT\_SUPE, [1032](#)  
SBC\_UJA\_GL\_EVNT\_STAT\_SUPE\_NO, [1032](#)  
SBC\_UJA\_GL\_EVNT\_STAT\_SYSE, [1032](#)  
SBC\_UJA\_GL\_EVNT\_STAT\_SYSE\_NO, [1032](#)  
SBC\_UJA\_GL\_EVNT\_STAT\_TRXE, [1032](#)  
SBC\_UJA\_GL\_EVNT\_STAT\_TRXE\_NO, [1032](#)  
SBC\_UJA\_GL\_EVNT\_STAT\_WPE, [1033](#)  
SBC\_UJA\_GL\_EVNT\_STAT\_WPE\_NO, [1033](#)  
SBC\_UJA\_IDENTIF, [1036](#)  
SBC\_UJA\_IDENTIF\_0, [1035](#)  
SBC\_UJA\_IDENTIF\_1, [1035](#)  
SBC\_UJA\_IDENTIF\_2, [1035](#)  
SBC\_UJA\_IDENTIF\_3, [1035](#)  
SBC\_UJA\_LOCK, [1035](#)  
SBC\_UJA\_MAIN, [1035](#)  
SBC\_UJA\_MAIN\_NMS\_NORMAL, [1033](#)  
SBC\_UJA\_MAIN\_NMS\_PWR\_UP, [1033](#)  
SBC\_UJA\_MAIN\_OTWS\_ABOVE, [1033](#)  
SBC\_UJA\_MAIN\_OTWS\_BELOW, [1033](#)  
SBC\_UJA\_MAIN\_RSS\_CAN\_WAKEUP, [1034](#)  
SBC\_UJA\_MAIN\_RSS\_DIAG\_WAKEUP, [1034](#)  
SBC\_UJA\_MAIN\_RSS\_ILLEG\_SLP, [1034](#)  
SBC\_UJA\_MAIN\_RSS\_ILLEG\_WATCH, [1034](#)  
SBC\_UJA\_MAIN\_RSS\_LFT\_OVERTM, [1034](#)  
SBC\_UJA\_MAIN\_RSS\_OFF\_MODE, [1034](#)  
SBC\_UJA\_MAIN\_RSS\_OVF\_SLP, [1034](#)  
SBC\_UJA\_MAIN\_RSS\_RSTN\_PULDW, [1034](#)  
SBC\_UJA\_MAIN\_RSS\_SLP\_WAKEUP, [1034](#)  
SBC\_UJA\_MAIN\_RSS\_V1\_UNDERV, [1034](#)  
SBC\_UJA\_MAIN\_RSS\_WAKE\_SLP, [1034](#)  
SBC\_UJA\_MAIN\_RSS\_WATCH\_OVF, [1034](#)  
SBC\_UJA\_MAIN\_RSS\_WATCH\_TRIG, [1034](#)  
SBC\_UJA\_MASK\_0, [1035](#)  
SBC\_UJA\_MASK\_1, [1035](#)  
SBC\_UJA\_MASK\_2, [1035](#)  
SBC\_UJA\_MASK\_3, [1035](#)  
SBC\_UJA\_MEMORY\_0, [1035](#)  
SBC\_UJA\_MEMORY\_1, [1035](#)  
SBC\_UJA\_MEMORY\_2, [1035](#)  
SBC\_UJA\_MEMORY\_3, [1035](#)  
SBC\_UJA\_MODE, [1035](#)  
SBC\_UJA\_MODE\_MC\_NORMAL, [1034](#)  
SBC\_UJA\_MODE\_MC\_SLEEP, [1034](#)  
SBC\_UJA\_MODE\_MC\_STANDBY, [1034](#)  
SBC\_UJA\_MTPNV\_CRC, [1036](#)  
SBC\_UJA\_MTPNV\_STAT, [1036](#)  
SBC\_UJA\_MTPNV\_STAT\_ECCS, [1034](#)  
SBC\_UJA\_MTPNV\_STAT\_ECCS\_NO, [1034](#)  
SBC\_UJA\_MTPNV\_STAT\_NVMPs, [1034](#)  
SBC\_UJA\_MTPNV\_STAT\_NVMPs\_NO, [1034](#)  
SBC\_UJA\_REGULATOR, [1035](#)  
SBC\_UJA\_REGULATOR\_PDC\_HV, [1036](#)  
SBC\_UJA\_REGULATOR\_PDC\_LV, [1036](#)  
SBC\_UJA\_REGULATOR\_V1RTC\_60, [1036](#)  
SBC\_UJA\_REGULATOR\_V1RTC\_70, [1036](#)  
SBC\_UJA\_REGULATOR\_V1RTC\_80, [1036](#)  
SBC\_UJA\_REGULATOR\_V1RTC\_90, [1036](#)  
SBC\_UJA\_REGULATOR\_V2C\_N, [1036](#)  
SBC\_UJA\_REGULATOR\_V2C\_N\_S\_R, [1036](#)  
SBC\_UJA\_REGULATOR\_V2C\_N\_S\_S\_R, [1036](#)  
SBC\_UJA\_REGULATOR\_V2C\_OFF, [1036](#)  
SBC\_UJA\_SBC, [1036](#)  
SBC\_UJA\_SBC\_FNMC\_DIS, [1037](#)  
SBC\_UJA\_SBC\_FNMC\_EN, [1037](#)  
SBC\_UJA\_SBC\_SDMC\_DIS, [1037](#)  
SBC\_UJA\_SBC\_SDMC\_EN, [1037](#)  
SBC\_UJA\_SBC\_SLPC\_AC, [1037](#)  
SBC\_UJA\_SBC\_SLPC\_IG, [1037](#)  
SBC\_UJA\_SBC\_V1RTSUC\_60, [1037](#)  
SBC\_UJA\_SBC\_V1RTSUC\_70, [1037](#)  
SBC\_UJA\_SBC\_V1RTSUC\_80, [1037](#)  
SBC\_UJA\_SBC\_V1RTSUC\_90, [1037](#)  
SBC\_UJA\_START\_UP, [1036](#)  
SBC\_UJA\_START\_UP\_RLC\_01\_01p5, [1038](#)  
SBC\_UJA\_START\_UP\_RLC\_03p6\_05, [1038](#)  
SBC\_UJA\_START\_UP\_RLC\_10\_12p5, [1038](#)  
SBC\_UJA\_START\_UP\_RLC\_20\_25p0, [1038](#)  
SBC\_UJA\_START\_UP\_V2SUC\_00, [1038](#)  
SBC\_UJA\_START\_UP\_V2SUC\_11, [1038](#)  
SBC\_UJA\_SUP\_EVNT\_STAT, [1035](#)  
SBC\_UJA\_SUP\_EVNT\_STAT\_V1U, [1038](#)  
SBC\_UJA\_SUP\_EVNT\_STAT\_V1U\_NO, [1038](#)  
SBC\_UJA\_SUP\_EVNT\_STAT\_V2O, [1038](#)  
SBC\_UJA\_SUP\_EVNT\_STAT\_V2O\_NO, [1038](#)  
SBC\_UJA\_SUP\_EVNT\_STAT\_V2U, [1038](#)  
SBC\_UJA\_SUP\_EVNT\_STAT\_V2U\_NO, [1038](#)  
SBC\_UJA\_SUPPLY\_EVNT, [1035](#)

- SBC\_UJA\_SUPPLY\_EVNT\_V1UE\_DIS, [1039](#)
- SBC\_UJA\_SUPPLY\_EVNT\_V1UE\_EN, [1039](#)
- SBC\_UJA\_SUPPLY\_EVNT\_V2OE\_DIS, [1039](#)
- SBC\_UJA\_SUPPLY\_EVNT\_V2OE\_EN, [1039](#)
- SBC\_UJA\_SUPPLY\_EVNT\_V2UE\_DIS, [1039](#)
- SBC\_UJA\_SUPPLY\_EVNT\_V2UE\_EN, [1039](#)
- SBC\_UJA\_SUPPLY\_STAT, [1035](#)
- SBC\_UJA\_SUPPLY\_STAT\_V1S\_VAB, [1039](#)
- SBC\_UJA\_SUPPLY\_STAT\_V1S\_VBE, [1039](#)
- SBC\_UJA\_SUPPLY\_STAT\_V2S\_DIS, [1039](#)
- SBC\_UJA\_SUPPLY\_STAT\_V2S\_VAB, [1039](#)
- SBC\_UJA\_SUPPLY\_STAT\_V2S\_VBE, [1039](#)
- SBC\_UJA\_SUPPLY\_STAT\_V2S\_VOK, [1039](#)
- SBC\_UJA\_SYS\_EVNT\_OTWE\_DIS, [1040](#)
- SBC\_UJA\_SYS\_EVNT\_OTWE\_EN, [1040](#)
- SBC\_UJA\_SYS\_EVNT\_SPIFE\_DIS, [1040](#)
- SBC\_UJA\_SYS\_EVNT\_SPIFE\_EN, [1040](#)
- SBC\_UJA\_SYS\_EVNT\_STAT, [1035](#)
- SBC\_UJA\_SYS\_EVNT\_STAT\_OTW, [1040](#)
- SBC\_UJA\_SYS\_EVNT\_STAT\_OTW\_NO, [1040](#)
- SBC\_UJA\_SYS\_EVNT\_STAT\_PO, [1040](#)
- SBC\_UJA\_SYS\_EVNT\_STAT\_PO\_NO, [1040](#)
- SBC\_UJA\_SYS\_EVNT\_STAT\_SPIF, [1040](#)
- SBC\_UJA\_SYS\_EVNT\_STAT\_SPIF\_NO, [1040](#)
- SBC\_UJA\_SYS\_EVNT\_STAT\_WDF, [1041](#)
- SBC\_UJA\_SYS\_EVNT\_STAT\_WDF\_NO, [1041](#)
- SBC\_UJA\_SYSTEM\_EVNT, [1035](#)
- SBC\_UJA\_TIMEOUT, [1029](#)
- SBC\_UJA\_TRANS\_EVNT, [1035](#)
- SBC\_UJA\_TRANS\_EVNT\_CBSE\_DIS, [1041](#)
- SBC\_UJA\_TRANS\_EVNT\_CBSE\_EN, [1041](#)
- SBC\_UJA\_TRANS\_EVNT\_CFE\_DIS, [1041](#)
- SBC\_UJA\_TRANS\_EVNT\_CFE\_EN, [1041](#)
- SBC\_UJA\_TRANS\_EVNT\_CWE\_DIS, [1041](#)
- SBC\_UJA\_TRANS\_EVNT\_CWE\_EN, [1041](#)
- SBC\_UJA\_TRANS\_EVNT\_STAT, [1035](#)
- SBC\_UJA\_TRANS\_EVNT\_STAT\_CBS, [1041](#)
- SBC\_UJA\_TRANS\_EVNT\_STAT\_CBS\_NO, [1041](#)
- SBC\_UJA\_TRANS\_EVNT\_STAT\_CF, [1042](#)
- SBC\_UJA\_TRANS\_EVNT\_STAT\_CF\_NO, [1042](#)
- SBC\_UJA\_TRANS\_EVNT\_STAT\_CW, [1042](#)
- SBC\_UJA\_TRANS\_EVNT\_STAT\_CW\_NO, [1042](#)
- SBC\_UJA\_TRANS\_EVNT\_STAT\_PNFDE, [1042](#)
- SBC\_UJA\_TRANS\_EVNT\_STAT\_PNFDE\_NO, [1042](#)
- SBC\_UJA\_TRANS\_STAT, [1035](#)
- SBC\_UJA\_TRANS\_STAT\_CBSS\_ACT, [1042](#)
- SBC\_UJA\_TRANS\_STAT\_CBSS\_INACT, [1042](#)
- SBC\_UJA\_TRANS\_STAT\_CFS\_NO\_TXD, [1043](#)
- SBC\_UJA\_TRANS\_STAT\_CFS\_TXD, [1043](#)
- SBC\_UJA\_TRANS\_STAT\_COSCS\_NRUN, [1043](#)
- SBC\_UJA\_TRANS\_STAT\_COSCS\_RUN, [1043](#)
- SBC\_UJA\_TRANS\_STAT\_CPNERR\_DET, [1043](#)
- SBC\_UJA\_TRANS\_STAT\_CPNERR\_NO\_DET, [1043](#)
- SBC\_UJA\_TRANS\_STAT\_CPNS\_ERR, [1043](#)
- SBC\_UJA\_TRANS\_STAT\_CPNS\_OK, [1043](#)
- SBC\_UJA\_TRANS\_STAT\_CTS\_ACT, [1043](#)
- SBC\_UJA\_TRANS\_STAT\_CTS\_INACT, [1043](#)
- SBC\_UJA\_TRANS\_STAT\_VCS\_AB, [1044](#)
- SBC\_UJA\_TRANS\_STAT\_VCS\_BE, [1044](#)
- SBC\_UJA\_WAKE\_EN, [1035](#)
- SBC\_UJA\_WAKE\_EN\_WPFE\_DIS, [1044](#)
- SBC\_UJA\_WAKE\_EN\_WPFE\_EN, [1044](#)
- SBC\_UJA\_WAKE\_EN\_WPRE\_DIS, [1044](#)
- SBC\_UJA\_WAKE\_EN\_WPRE\_EN, [1044](#)
- SBC\_UJA\_WAKE\_EVNT\_STAT, [1036](#)
- SBC\_UJA\_WAKE\_EVNT\_STAT\_WPF, [1044](#)
- SBC\_UJA\_WAKE\_EVNT\_STAT\_WPF\_NO, [1044](#)
- SBC\_UJA\_WAKE\_EVNT\_STAT\_WPR, [1044](#)
- SBC\_UJA\_WAKE\_EVNT\_STAT\_WPR\_NO, [1044](#)
- SBC\_UJA\_WAKE\_STAT, [1035](#)
- SBC\_UJA\_WAKE\_STAT\_WPVS\_AB, [1045](#)
- SBC\_UJA\_WAKE\_STAT\_WPVS\_BE, [1045](#)
- SBC\_UJA\_WTDOG\_CTR, [1035](#)
- SBC\_UJA\_WTDOG\_CTR\_NWP\_1024, [1045](#)
- SBC\_UJA\_WTDOG\_CTR\_NWP\_128, [1045](#)
- SBC\_UJA\_WTDOG\_CTR\_NWP\_16, [1045](#)
- SBC\_UJA\_WTDOG\_CTR\_NWP\_256, [1045](#)
- SBC\_UJA\_WTDOG\_CTR\_NWP\_32, [1045](#)
- SBC\_UJA\_WTDOG\_CTR\_NWP\_4096, [1045](#)
- SBC\_UJA\_WTDOG\_CTR\_NWP\_64, [1045](#)
- SBC\_UJA\_WTDOG\_CTR\_NWP\_8, [1045](#)
- SBC\_UJA\_WTDOG\_CTR\_WMC\_AUTO, [1045](#)
- SBC\_UJA\_WTDOG\_CTR\_WMC\_TIME, [1045](#)
- SBC\_UJA\_WTDOG\_CTR\_WMC\_WIND, [1045](#)
- SBC\_UJA\_WTDOG\_STAT, [1035](#)
- SBC\_UJA\_WTDOG\_STAT\_FNMS\_N\_NORMAL, [1046](#)
- SBC\_UJA\_WTDOG\_STAT\_FNMS\_NORMA←L, [1046](#)
- SBC\_UJA\_WTDOG\_STAT\_SDMS\_N\_NORMAL, [1046](#)
- SBC\_UJA\_WTDOG\_STAT\_SDMS\_NORMA←L, [1046](#)
- SBC\_UJA\_WTDOG\_STAT\_WDS\_FIH, [1046](#)
- SBC\_UJA\_WTDOG\_STAT\_WDS\_OFF, [1046](#)
- SBC\_UJA\_WTDOG\_STAT\_WDS\_SEH, [1046](#)
- sbcan\_cfdc\_t, [1030](#)
- sbcan\_cmc\_t, [1030](#)
- sbcan\_cpnc\_t, [1030](#)
- sbcan\_pncok\_t, [1031](#)
- sbdat\_rate\_t, [1031](#)
- sbdata\_mask\_t, [1029](#)
- sbfail\_safe\_lhc\_t, [1031](#)
- sbfail\_safe\_rcc\_t, [1029](#)
- sbframe\_ctr\_dlc\_t, [1029](#)
- sbframe\_ctr\_ide\_t, [1031](#)
- sbframe\_ctr\_pndm\_t, [1032](#)
- sbgl\_evnt\_stat\_supe\_t, [1032](#)
- sbgl\_evnt\_stat\_syse\_t, [1032](#)
- sbgl\_evnt\_stat\_trxe\_t, [1032](#)
- sbgl\_evnt\_stat\_wpe\_t, [1032](#)
- sbidentif\_mask\_t, [1030](#)
- sbidentifier\_t, [1030](#)
- sblock\_t, [1033](#)

- sbc\_main\_nms\_t, [1033](#)
- sbc\_main\_otws\_t, [1033](#)
- sbc\_main\_rss\_t, [1033](#)
- sbc\_mode\_mc\_t, [1034](#)
- sbc\_mtpnv\_stat\_eccs\_t, [1034](#)
- sbc\_mtpnv\_stat\_nvmps\_t, [1034](#)
- sbc\_mtpnv\_stat\_wrcnts\_t, [1030](#)
- sbc\_register\_t, [1034](#)
- sbc\_regulator\_pdc\_t, [1036](#)
- sbc\_regulator\_v1rtc\_t, [1036](#)
- sbc\_regulator\_v2c\_t, [1036](#)
- sbc\_sbc\_fnmc\_t, [1036](#)
- sbc\_sbc\_sdmc\_t, [1037](#)
- sbc\_sbc\_slpc\_t, [1037](#)
- sbc\_sbc\_v1rtsuc\_t, [1037](#)
- sbc\_start\_up\_rlc\_t, [1037](#)
- sbc\_start\_up\_v2suc\_t, [1038](#)
- sbc\_sup\_evnt\_stat\_v1u\_t, [1038](#)
- sbc\_sup\_evnt\_stat\_v2o\_t, [1038](#)
- sbc\_sup\_evnt\_stat\_v2u\_t, [1038](#)
- sbc\_supply\_evnt\_v1ue\_t, [1038](#)
- sbc\_supply\_evnt\_v2oe\_t, [1039](#)
- sbc\_supply\_evnt\_v2ue\_t, [1039](#)
- sbc\_supply\_stat\_v1s\_t, [1039](#)
- sbc\_supply\_stat\_v2s\_t, [1039](#)
- sbc\_sys\_evnt\_otwe\_t, [1039](#)
- sbc\_sys\_evnt\_spife\_t, [1040](#)
- sbc\_sys\_evnt\_stat\_otw\_t, [1040](#)
- sbc\_sys\_evnt\_stat\_po\_t, [1040](#)
- sbc\_sys\_evnt\_stat\_spif\_t, [1040](#)
- sbc\_sys\_evnt\_stat\_wdf\_t, [1040](#)
- sbc\_trans\_evnt\_cbse\_t, [1041](#)
- sbc\_trans\_evnt\_cfe\_t, [1041](#)
- sbc\_trans\_evnt\_cwe\_t, [1041](#)
- sbc\_trans\_evnt\_stat\_cbs\_t, [1041](#)
- sbc\_trans\_evnt\_stat\_cf\_t, [1042](#)
- sbc\_trans\_evnt\_stat\_cw\_t, [1042](#)
- sbc\_trans\_evnt\_stat\_pnfde\_t, [1042](#)
- sbc\_trans\_stat\_cbss\_t, [1042](#)
- sbc\_trans\_stat\_cfs\_t, [1042](#)
- sbc\_trans\_stat\_coscs\_t, [1043](#)
- sbc\_trans\_stat\_cpnrerr\_t, [1043](#)
- sbc\_trans\_stat\_cpns\_t, [1043](#)
- sbc\_trans\_stat\_cts\_t, [1043](#)
- sbc\_trans\_stat\_vcs\_t, [1043](#)
- sbc\_wake\_en\_wpfe\_t, [1044](#)
- sbc\_wake\_en\_wpre\_t, [1044](#)
- sbc\_wake\_evnt\_stat\_wpf\_t, [1044](#)
- sbc\_wake\_evnt\_stat\_wpr\_t, [1044](#)
- sbc\_wake\_stat\_wpvs\_t, [1044](#)
- sbc\_wtdog\_ctr\_nwp\_t, [1045](#)
- sbc\_wtdog\_ctr\_wmc\_t, [1045](#)
- sbc\_wtdog\_stat\_fnms\_t, [1045](#)
- sbc\_wtdog\_stat\_sdms\_t, [1046](#)
- sbc\_wtdog\_stat\_wds\_t, [1046](#)
- uart\_bit\_count\_per\_char\_t
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1053](#)
- uart\_instance\_t, [1089](#)
  - instIdx, [1090](#)
  - instType, [1090](#)
- uart\_parity\_mode\_t
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)
- uart\_stop\_bit\_count\_t
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)
- uart\_transfer\_type\_t
  - Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1054](#)
- uart\_user\_config\_t, [1052](#)
  - baudRate, [1052](#)
  - bitCount, [1052](#)
  - extension, [1052](#)
  - parityMode, [1052](#)
  - rxCallback, [1052](#)
  - rxCallbackParam, [1052](#)
  - rxDMAChannel, [1052](#)
  - stopBitCount, [1053](#)
  - transferType, [1053](#)
  - txCallback, [1053](#)
  - txCallbackParam, [1053](#)
  - txDMAChannel, [1053](#)
- uja113x\_bmc\_cfg\_t, [165](#)
  - bmOverVoltThr, [166](#)
  - bmOverVoltThrRel, [166](#)
  - bmSrcCtrl, [166](#)
  - bmUndrVoltThr, [166](#)
  - bmUndrVoltThrRel, [166](#)
- uja113x\_bmetsc\_bmsc\_t
  - 02. Battery monitor configuration, [167](#)
- uja113x\_bmhc\_bmhoc\_t
  - 02. Battery monitor configuration, [167](#)
- uja113x\_bmhc\_bmhuc\_t
  - 02. Battery monitor configuration, [167](#)
- uja113x\_bmotc\_bmotc\_t
  - 02. Battery monitor configuration, [167](#)
- uja113x\_bmutc\_bmutc\_t
  - 02. Battery monitor configuration, [167](#)
- uja113x\_boldt\_cfg\_t, [188](#)
  - io15OpenLoadThrCtrl, [189](#)
  - io26OpenLoadThrCtrl, [189](#)
  - io37OpenLoadThrCtrl, [189](#)
  - io48OpenLoadThrCtrl, [189](#)
- uja113x\_boldt\_ionoltc\_t
  - 02. Battery monitor configuration, [167](#)
- uja113x\_bscdt\_cfg\_t, [188](#)
  - io15ShortCircuitThrCtrl, [188](#)
  - io26ShortCircuitThrCtrl, [188](#)
  - io37ShortCircuitThrCtrl, [188](#)
  - io48ShortCircuitThrCtrl, [188](#)
- uja113x\_bscdt\_ionsctc\_t
  - 02. Battery monitor configuration, [168](#)
- uja113x\_bwtc\_bnwtc\_t
  - 04. HVIO bank 0 & 1 control, [189](#)
- uja113x\_canc\_cfdc\_t



- 01. CAN & CAN partial networking configuration, 154
- uja113x\_canc\_cfg\_t, 151
  - canFDCtrl, 151
  - canModeCtrl, 151
  - canPartNetCtrl, 151
  - canShutdownCtrl, 152
- uja113x\_canc\_cmc\_t
  - 01. CAN & CAN partial networking configuration, 155
- uja113x\_canc\_cpnc\_t
  - 01. CAN & CAN partial networking configuration, 154
- uja113x\_canc\_csc\_t
  - 01. CAN & CAN partial networking configuration, 155
- uja113x\_chip\_t
  - 00. Library internals, 148
- uja113x\_cpnc\_cfg\_t, 152
  - canDataRate, 152
  - dataLenCode, 152
  - id, 152
  - idMask, 152
  - ide, 152
  - partNetDataMask, 153
- uja113x\_dm\_cfg\_t, 153
  - dm0, 153
  - dm1, 153
  - dm2, 153
  - dm3, 153
  - dm4, 154
  - dm5, 154
  - dm6, 154
  - dm7, 154
- uja113x\_dr\_cdr\_t
  - 01. CAN & CAN partial networking configuration, 155
- uja113x\_driver\_t, 147
  - chipType, 147
  - lpsplInstance, 147
  - timeout, 147
- uja113x\_drv\_cfg\_t, 142
  - chipType, 142
  - lpsplInstance, 142
  - timeout, 143
- uja113x\_fc\_dlc\_t
  - 01. CAN & CAN partial networking configuration, 156
- uja113x\_fc\_ide\_t
  - 01. CAN & CAN partial networking configuration, 154
- uja113x\_fc\_pndm\_t
  - 01. CAN & CAN partial networking configuration, 154
- uja113x\_fsc\_cfg\_t, 179
  - enCtrl, 179
  - enDriverCtrl, 179
  - enShutdownCtrl, 179
  - limpHomeCtrl, 179
  - resetCntCtrl, 179
- uja113x\_fsc\_enc\_t
  - 03. Fail-safe configuration, 180
- uja113x\_fsc\_endc\_t
  - 03. Fail-safe configuration, 180
- uja113x\_fsc\_ensc\_t
  - 03. Fail-safe configuration, 180
- uja113x\_fsc\_lhc\_t
  - 03. Fail-safe configuration, 180
- uja113x\_fsc\_rcc\_t
  - 03. Fail-safe configuration, 180
- uja113x\_hvio\_bank\_cfg\_t, 143
  - bankFailIntMask, 143
  - bankWupIntMask, 143
  - bankWupThr, 143
  - boldtBankOpenLoad, 143
  - bscdtBankShortCirc, 143
  - hvioConfig, 144
- uja113x\_hvio\_cfg\_t, 187
  - ionActivateCtrl, 188
  - ionConfCtrl, 188
  - ionShutdownCtrl, 188
- uja113x\_hvion\_ionac\_t
  - 04. HVIO bank 0 & 1 control, 189
- uja113x\_hvion\_ioncc\_t
  - 04. HVIO bank 0 & 1 control, 190
- uja113x\_hvion\_ionsc\_t
  - 04. HVIO bank 0 & 1 control, 190
- uja113x\_id\_id\_t
  - 01. CAN & CAN partial networking configuration, 155
- uja113x\_idm\_idm\_t
  - 01. CAN & CAN partial networking configuration, 155
- uja113x\_init\_cfg\_t, 144
  - bmcBatMonitorCfg, 145
  - cancCanCfg, 145
  - cpncCanPartNetCfg, 145
  - dmCanDataMaskCfg, 145
  - fscCfg, 145
  - hvioBank0Cfg, 145
  - hvioBank1Cfg, 145
  - lckcLockRegCfg, 145
  - linxLinCtrl, 146
  - mcSbcMode, 146
  - memory0, 146
  - memory1, 146
  - memory2, 146
  - memory3, 146
  - rcRegulatorCfg, 146
  - smpscSmpsCfg, 146
  - supieMask, 146
  - sysieMask, 146
  - tieMask, 146
  - timscTimer1Cfg, 146
  - timscTimer2Cfg, 147
  - timscTimer3Cfg, 147

- timscTimer4Cfg, [147](#)
- wdcWatchdogCfg, [147](#)
- uja113x\_lckc\_cfg\_t, [193](#)
  - lock0, [194](#)
  - lock1, [194](#)
  - lock2, [194](#)
  - lock3, [194](#)
  - lock4, [194](#)
  - lock5, [194](#)
  - lock6, [194](#)
- uja113x\_lckc\_lkxc\_t
  - 04. Lock control, [195](#)
- uja113x\_linc\_cfg\_t, [169](#)
  - lin1ModeCtrl, [170](#)
  - lin1SlopeCtrl, [170](#)
  - lin2ModeCtrl, [170](#)
  - lin2SlopeCtrl, [170](#)
- uja113x\_linc\_lmc1\_t
  - 02. LIN configuration, [170](#)
- uja113x\_linc\_lmc2\_t
  - 02. LIN configuration, [170](#)
- uja113x\_linc\_lsc1\_t
  - 02. LIN configuration, [171](#)
- uja113x\_linc\_lsc2\_t
  - 02. LIN configuration, [171](#)
- uja113x\_mc\_mc\_t
  - 01. Mode handling, [159](#)
- uja113x\_mtpnv\_cfg\_t, [207](#)
  - highSideCtrl, [208](#)
  - lowSideCtrl, [208](#)
  - sbcConfigCtrl, [208](#)
  - startupCtrl, [208](#)
- uja113x\_rc\_cfg\_t, [162](#)
  - v1RstThrCtrl, [163](#)
  - v2Ctrl, [163](#)
  - v2ShutdownCtrl, [163](#)
- uja113x\_rc\_v1rtc\_t
  - 01. Regulator configuration, [163](#)
- uja113x\_rc\_v2c\_t
  - 01. Regulator configuration, [163](#)
- uja113x\_rc\_v2sc\_t
  - 01. Regulator configuration, [163](#)
- uja113x\_smpsc\_cfg\_t, [182](#)
  - smpsCtrl, [183](#)
  - smpsOutVolt, [183](#)
  - smpsOvertempCtrl, [183](#)
- uja113x\_smpsc\_smpsc\_t
  - 03. SMPS configuration, [183](#)
- uja113x\_smpsc\_smpsotc\_t
  - 03. SMPS configuration, [183](#)
- uja113x\_smpsovc\_smpsoc\_t
  - 03. SMPS configuration, [183](#)
- uja113x\_timsc\_cfg\_t, [196](#)
  - nTimer, [197](#)
  - timXDutyCycleCtrl, [197](#)
  - timXModeCtrl, [197](#)
  - timXPeriodCtrl, [197](#)
- uja113x\_timxc\_txmc\_t
  - 05. Timer control, [197](#)
- uja113x\_timxc\_tnpc\_t
  - 05. Timer control, [198](#)
- uja113x\_timxdcc\_txdcc\_t
  - 05. Timer control, [197](#)
- uja113x\_wtdc\_cfg\_t, [175](#)
  - nomWtdPeriod, [175](#)
  - wtdModeCtrl, [175](#)
- uja113x\_wtdc\_nwp\_t
  - 02. Watchdog, [175](#)
- uja113x\_wtdc\_wmc\_t
  - 02. Watchdog, [176](#)
- Universal Asynchronous Receiver/Transmitter - Peripheral Abstraction Layer (UART PAL), [1047](#)
  - UART\_10\_BITS\_PER\_CHAR, [1054](#)
  - UART\_15\_BITS\_PER\_CHAR, [1054](#)
  - UART\_16\_BITS\_PER\_CHAR, [1054](#)
  - UART\_7\_BITS\_PER\_CHAR, [1054](#)
  - UART\_8\_BITS\_PER\_CHAR, [1054](#)
  - UART\_9\_BITS\_PER\_CHAR, [1054](#)
  - UART\_AbortReceivingData, [1054](#)
  - UART\_AbortSendingData, [1055](#)
  - UART\_Deinit, [1055](#)
  - UART\_GetBaudRate, [1055](#)
  - UART\_GetDefaultConfig, [1055](#)
  - UART\_GetReceiveStatus, [1056](#)
  - UART\_GetTransmitStatus, [1056](#)
  - UART\_Init, [1057](#)
  - UART\_ONE\_STOP\_BIT, [1054](#)
  - UART\_PARITY\_DISABLED, [1054](#)
  - UART\_PARITY\_EVEN, [1054](#)
  - UART\_PARITY\_ODD, [1054](#)
  - UART\_ReceiveData, [1057](#)
  - UART\_ReceiveDataBlocking, [1057](#)
  - UART\_SendData, [1058](#)
  - UART\_SendDataBlocking, [1058](#)
  - UART\_SetBaudRate, [1058](#)
  - UART\_SetRxBuffer, [1059](#)
  - UART\_SetTxBuffer, [1059](#)
  - UART\_TWO\_STOP\_BIT, [1054](#)
  - UART\_USING\_DMA, [1054](#)
  - UART\_USING\_INTERRUPTS, [1054](#)
  - uart\_bit\_count\_per\_char\_t, [1053](#)
  - uart\_parity\_mode\_t, [1054](#)
  - uart\_stop\_bit\_count\_t, [1054](#)
  - uart\_transfer\_type\_t, [1054](#)
- updateEnable
  - rtc\_init\_config\_t, [896](#)
  - wdog\_user\_config\_t, [1072](#)
- User provided call-outs, [1060](#)
  - I\_sys\_irq\_disable, [1060](#)
  - I\_sys\_irq\_restore, [1060](#)
- userGain
  - adc\_calibration\_t, [219](#)
- userOffset
  - adc\_calibration\_t, [219](#)
- V1RTC\_60\_NOMINAL\_OUT
  - 01. Regulator configuration, [163](#)

- V1RTC\_70\_NOMINAL\_OUT
  - 01. Regulator configuration, [163](#)
- V1RTC\_80\_NOMINAL\_OUT
  - 01. Regulator configuration, [163](#)
- V1RTC\_90\_NOMINAL\_OUT
  - 01. Regulator configuration, [163](#)
- V1RTSUC\_60\_NOM
  - SBC configuration control options (0x74), [930](#)
- V1RTSUC\_70\_NOM
  - SBC configuration control options (0x74), [930](#)
- V1RTSUC\_80\_NOM
  - SBC configuration control options (0x74), [930](#)
- V1RTSUC\_90\_NOM
  - SBC configuration control options (0x74), [930](#)
- v1RstThrCtrl
  - uja113x\_rc\_cfg\_t, [163](#)
- V1UIE\_DISABLE
  - Supply interrupt enable configuration options (0x1C), [976](#)
- V1UIE\_ENABLE
  - Supply interrupt enable configuration options (0x1C), [976](#)
- v1rtc
  - sbc\_regulator\_t, [1015](#)
- v1rtsuc
  - sbc\_sbc\_t, [1014](#)
- v1s
  - sbc\_supply\_status\_t, [1023](#)
- v1u
  - sbc\_sup\_evnt\_stat\_t, [1025](#)
- v1ue
  - sbc\_supply\_evnt\_t, [1015](#)
- V2C\_OFF
  - 01. Regulator configuration, [163](#)
- V2C\_ON\_N
  - 01. Regulator configuration, [163](#)
- V2C\_ON\_NSR
  - 01. Regulator configuration, [163](#)
- V2C\_ON\_NSSRF
  - 01. Regulator configuration, [163](#)
- v2Ctrl
  - uja113x\_rc\_cfg\_t, [163](#)
- V2SC\_FULL
  - 01. Regulator configuration, [164](#)
- V2SC\_NONE
  - 01. Regulator configuration, [163](#)
- V2SC\_OVERVOLTAGE
  - 01. Regulator configuration, [163](#)
- V2SC\_UNDRVOLTAGE
  - 01. Regulator configuration, [163](#)
- V2SUC\_ALWAYS\_OFF
  - Start-up control configuration options (0x73), [972](#)
- V2SUC\_ALWAYS\_ON
  - Start-up control configuration options (0x73), [972](#)
- v2ShutdownCtrl
  - uja113x\_rc\_cfg\_t, [163](#)
- v2c
  - sbc\_regulator\_t, [1015](#)
- v2o
  - sbc\_sup\_evnt\_stat\_t, [1025](#)
- v2oe
  - sbc\_supply\_evnt\_t, [1016](#)
- v2s
  - sbc\_supply\_status\_t, [1023](#)
- v2suc
  - sbc\_start\_up\_t, [1015](#)
- v2u
  - sbc\_sup\_evnt\_stat\_t, [1025](#)
- v2ue
  - sbc\_supply\_evnt\_t, [1016](#)
- VBADC1\_ADDRESS
  - Register address map, [926](#)
- VBADC2\_ADDRESS
  - Register address map, [926](#)
- VEXTAC\_NO\_PROTECTION
  - SBC configuration control options (0x74), [931](#)
- VEXTAC\_SENSOR\_SUPPLY
  - SBC configuration control options (0x74), [931](#)
- VEXTOIE\_DISABLE
  - Supply interrupt enable configuration options (0x1C), [976](#)
- VEXTOIE\_ENABLE
  - Supply interrupt enable configuration options (0x1C), [976](#)
- VEXTUIE\_DISABLE
  - Supply interrupt enable configuration options (0x1C), [977](#)
- VEXTUIE\_ENABLE
  - Supply interrupt enable configuration options (0x1C), [977](#)
- VLPR\_MODE
  - Clock Manager Driver, [305](#)
- VLPS\_MODE
  - Clock Manager Driver, [305](#)
- variant
  - lin\_product\_id\_t, [1085](#)
- vccrConfig
  - scg\_clock\_mode\_config\_t, [294](#)
- vcs
  - sbc\_trans\_stat\_t, [1024](#)
- verifStatus
  - csec\_state\_t, [262](#)
- virtChn
  - edma\_chn\_state\_t, [377](#)
- virtChnConfig
  - edma\_channel\_config\_t, [378](#)
- virtChnState
  - edma\_state\_t, [379](#)
- vlpProt
  - smc\_power\_mode\_protection\_config\_t, [878](#)
- voltage
  - cmp\_dac\_t, [330](#)
- voltageRef
  - adc\_converter\_config\_t, [217](#)
  - extension\_adc\_s32k1xx\_t, [237](#)
- voltageReferenceSource



- cmp\_dac\_t, [330](#)
- WDG PAL, [1061](#)
  - WDG\_ClearIntFlag, [1064](#)
  - WDG\_Deinit, [1064](#)
  - WDG\_GetCounter, [1065](#)
  - WDG\_GetDefaultConfig, [1065](#)
  - WDG\_IN\_ASSERT\_DISABLED, [1064](#)
  - WDG\_IN\_ASSERT\_ON\_LOGIC\_ONE, [1064](#)
  - WDG\_IN\_ASSERT\_ON\_LOGIC\_ZERO, [1064](#)
  - WDG\_Init, [1065](#)
  - WDG\_PAL\_BUS\_CLOCK, [1064](#)
  - WDG\_PAL\_LPO\_CLOCK, [1064](#)
  - WDG\_PAL\_SIRC\_CLOCK, [1064](#)
  - WDG\_PAL\_SOSC\_CLOCK, [1064](#)
  - WDG\_Refresh, [1066](#)
  - WDG\_SetInt, [1066](#)
  - WDG\_SetTimeout, [1066](#)
  - WDG\_SetWindow, [1066](#)
  - wdg\_clock\_source\_t, [1063](#)
  - wdg\_in\_assert\_logic\_t, [1064](#)
  - wdg\_inst\_type\_t, [1064](#)
- WDG\_ClearIntFlag
  - WDG PAL, [1064](#)
- WDG\_Deinit
  - WDG PAL, [1064](#)
- WDG\_GetCounter
  - WDG PAL, [1065](#)
- WDG\_GetDefaultConfig
  - WDG PAL, [1065](#)
- WDG\_IN\_ASSERT\_DISABLED
  - WDG PAL, [1064](#)
- WDG\_IN\_ASSERT\_ON\_LOGIC\_ONE
  - WDG PAL, [1064](#)
- WDG\_IN\_ASSERT\_ON\_LOGIC\_ZERO
  - WDG PAL, [1064](#)
- WDG\_Init
  - WDG PAL, [1065](#)
- WDG\_PAL\_BUS\_CLOCK
  - WDG PAL, [1064](#)
- WDG\_PAL\_LPO\_CLOCK
  - WDG PAL, [1064](#)
- WDG\_PAL\_SIRC\_CLOCK
  - WDG PAL, [1064](#)
- WDG\_PAL\_SOSC\_CLOCK
  - WDG PAL, [1064](#)
- WDG\_Refresh
  - WDG PAL, [1066](#)
- WDG\_SetInt
  - WDG PAL, [1066](#)
- WDG\_SetTimeout
  - WDG PAL, [1066](#)
- WDG\_SetWindow
  - WDG PAL, [1066](#)
- WDOG Driver, [1068](#)
  - WDOG\_BUS\_CLOCK, [1072](#)
  - WDOG\_DEBUG\_MODE, [1072](#)
  - WDOG\_DRV\_ClearIntFlag, [1073](#)
  - WDOG\_DRV\_Deinit, [1073](#)
  - WDOG\_DRV\_GetConfig, [1073](#)
  - WDOG\_DRV\_GetCounter, [1073](#)
  - WDOG\_DRV\_GetDefaultConfig, [1074](#)
  - WDOG\_DRV\_GetTestMode, [1075](#)
  - WDOG\_DRV\_SetInt, [1074](#)
  - WDOG\_DRV\_SetMode, [1075](#)
  - WDOG\_DRV\_SetTestMode, [1075](#)
  - WDOG\_DRV\_SetTimeout, [1075](#)
  - WDOG\_DRV\_SetWindow, [1076](#)
  - WDOG\_DRV\_Trigger, [1076](#)
  - WDOG\_LPO\_CLOCK, [1072](#)
  - WDOG\_SIRC\_CLOCK, [1072](#)
  - WDOG\_SOSC\_CLOCK, [1072](#)
  - WDOG\_STOP\_MODE, [1072](#)
  - WDOG\_TST\_DISABLED, [1073](#)
  - WDOG\_TST\_HIGH, [1073](#)
  - WDOG\_TST\_LOW, [1073](#)
  - WDOG\_TST\_USER, [1073](#)
  - WDOG\_WAIT\_MODE, [1072](#)
  - wdog\_clk\_source\_t, [1072](#)
  - wdog\_set\_mode\_t, [1072](#)
  - wdog\_test\_mode\_t, [1072](#)
- WDOG\_BUS\_CLOCK
  - WDOG Driver, [1072](#)
- WDOG\_DEBUG\_MODE
  - WDOG Driver, [1072](#)
- WDOG\_DRV\_ClearIntFlag
  - WDOG Driver, [1073](#)
- WDOG\_DRV\_Deinit
  - WDOG Driver, [1073](#)
- WDOG\_DRV\_GetConfig
  - WDOG Driver, [1073](#)
- WDOG\_DRV\_GetCounter
  - WDOG Driver, [1073](#)
- WDOG\_DRV\_GetDefaultConfig
  - WDOG Driver, [1074](#)
- WDOG\_DRV\_GetTestMode
  - WDOG Driver, [1074](#)
- WDOG\_DRV\_Init
  - WDOG Driver, [1074](#)
- WDOG\_DRV\_SetInt
  - WDOG Driver, [1074](#)
- WDOG\_DRV\_SetMode
  - WDOG Driver, [1075](#)
- WDOG\_DRV\_SetTestMode
  - WDOG Driver, [1075](#)
- WDOG\_DRV\_SetTimeout
  - WDOG Driver, [1075](#)
- WDOG\_DRV\_SetWindow
  - WDOG Driver, [1076](#)
- WDOG\_DRV\_Trigger
  - WDOG Driver, [1076](#)
- WDOG\_LPO\_CLOCK
  - WDOG Driver, [1072](#)
- WDOG\_SIRC\_CLOCK
  - WDOG Driver, [1072](#)
- WDOG\_SOSC\_CLOCK

- WDOG Driver, [1072](#)
- WDOG\_STOP\_MODE
  - WDOG Driver, [1072](#)
- WDOG\_TST\_DISABLED
  - WDOG Driver, [1073](#)
- WDOG\_TST\_HIGH
  - WDOG Driver, [1073](#)
- WDOG\_TST\_LOW
  - WDOG Driver, [1073](#)
- WDOG\_TST\_USER
  - WDOG Driver, [1073](#)
- WDOG\_WAIT\_MODE
  - WDOG Driver, [1072](#)
- WMC\_AUTONOMOUS
  - 02. Watchdog, [176](#)
- WMC\_TIMEOUT
  - 02. Watchdog, [176](#)
- WMC\_WINDOW
  - 02. Watchdog, [176](#)
- WTDC\_ADDRESS
  - Register address map, [926](#)
- WTDS\_ADDRESS
  - Register address map, [926](#)
- wait
  - wdg\_option\_mode\_t, [1062](#)
  - wdog\_op\_mode\_t, [1071](#)
- wakePin
  - sbc\_int\_config\_t, [1021](#)
  - sbc\_status\_group\_t, [1029](#)
- wakePinEvt
  - sbc\_evt\_capt\_t, [1027](#)
- watchdog
  - sbc\_int\_config\_t, [1021](#)
- Watchdog Peripheral Abstraction Layer (WDG PAL), [1077](#)
- Watchdog timer (WDOG), [1080](#)
- watchdogCtr
  - drv\_config\_t, [1083](#)
- wdcWatchdogCfg
  - uja113x\_init\_cfg\_t, [147](#)
- wdf
  - sbc\_sys\_evt\_stat\_t, [1025](#)
- wdg\_clock\_source\_t
  - WDG PAL, [1063](#)
- wdg\_config\_t, [1062](#)
  - clkSource, [1063](#)
  - extension, [1063](#)
  - intEnable, [1063](#)
  - opMode, [1063](#)
  - percentWindow, [1063](#)
  - prescalerEnable, [1063](#)
  - timeoutValue, [1063](#)
  - winEnable, [1063](#)
- wdg\_in\_assert\_logic\_t
  - WDG PAL, [1064](#)
- wdg\_inst\_type\_t
  - WDG PAL, [1064](#)
- wdg\_instance\_t, [1090](#)
- instIdx, [1090](#)
- instType, [1090](#)
- wdg\_option\_mode\_t, [1062](#)
  - debug, [1062](#)
  - stop, [1062](#)
  - wait, [1062](#)
- wdog\_clk\_source\_t
  - WDOG Driver, [1072](#)
- wdog\_op\_mode\_t, [1071](#)
  - debug, [1071](#)
  - stop, [1071](#)
  - wait, [1071](#)
- wdog\_set\_mode\_t
  - WDOG Driver, [1072](#)
- wdog\_test\_mode\_t
  - WDOG Driver, [1072](#)
- wdog\_user\_config\_t, [1071](#)
  - clkSource, [1071](#)
  - intEnable, [1071](#)
  - opMode, [1071](#)
  - prescalerEnable, [1072](#)
  - timeoutValue, [1072](#)
  - updateEnable, [1072](#)
  - winEnable, [1072](#)
  - windowValue, [1072](#)
- wds
  - sbc\_wtdog\_status\_t, [1022](#)
- whichPcs
  - lpspi\_master\_config\_t, [705](#)
  - lpspi\_slave\_config\_t, [709](#)
- winEnable
  - wdg\_config\_t, [1063](#)
  - wdog\_user\_config\_t, [1072](#)
- windowValue
  - wdog\_user\_config\_t, [1072](#)
- word\_status
  - lin\_protocol\_state\_t, [771](#)
- wordWidth
  - i2s\_user\_config\_t, [605](#)
- workMode
  - lptmr\_config\_t, [723](#)
- wpe
  - sbc\_gl\_evt\_stat\_t, [1024](#)
- wpf
  - sbc\_wake\_evt\_stat\_t, [1026](#)
- wpfe
  - sbc\_wake\_t, [1019](#)
- wpr
  - sbc\_wake\_evt\_stat\_t, [1026](#)
- wpre
  - sbc\_wake\_t, [1019](#)
- wrcnts
  - sbc\_mtpnv\_stat\_t, [1028](#)
- writeTranspose
  - crc\_user\_config\_t, [251](#)
- wsPin
  - extension\_flexio\_for\_i2s\_t, [606](#)
  - flexio\_i2s\_master\_user\_config\_t, [478](#)

- [flexio\\_i2s\\_slave\\_user\\_config\\_t](#), [480](#)
- [wtdModeCtrl](#)
  - [uja113x\\_wtdc\\_cfg\\_t](#), [175](#)
- [wtdog](#)
  - [sbc\\_status\\_group\\_t](#), [1029](#)
- [XOSC\\_EXT\\_REF](#)
  - [Clock Manager Driver](#), [310](#)
- [XOSC\\_INT\\_OSC](#)
  - [Clock Manager Driver](#), [310](#)
- [xorOut](#)
  - [crc\\_spec\\_t](#), [209](#)
- [xosc\\_ref\\_t](#)
  - [Clock Manager Driver](#), [310](#)
- [YEAR\\_RANGE\\_END](#)
  - [RTC Driver](#), [900](#)
- [YEAR\\_RANGE\\_START](#)
  - [RTC Driver](#), [900](#)
- [year](#)
  - [rtc\\_timedate\\_t](#), [895](#)